Зміст

					ДК61.403231.001 ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Сільчук В.І.			Пристрій для відображення реального часу з функцією календаря та будильника	Літ.	Арк. 1 / Аркушів
Перевірив		Корнєв В.П.					
Реценз.						КПІ ім. І. Сікорського, ФЕЛ, КЕОА, гр. ДК-61	
Н. Контр.							
Затвердив		Корнєв В.П.					

Перелік використаних скорочень

RTC	Real-time clock
МК	Мікроконтролер
GPIO	General purpose input-output
I2C	Inter-integrated circuit
LCD	Liquid crystal display
SDA	Serial data
SCK	Serial clock

Вступ

В сучасному світі — дуже стрімкому та насиченому, дуже важливою навичкою для людини є вміння правильно розраховувати, планувати та розподіляти свій час для виконання завдань, що виникають в процесі її життя — так званий «тайм-менеджмент», для того, щоб вчасно прибувати туди, куди потрібно, вдало планувати зустрічі та дотримуватись, при цьому, пунктуальності. Для точної синхронізації своїх планів з іншими членами соціуму, можна прив’язати вищенаведене планування до значень реального часу. Відповідно, в такому випадку, необхідний пристрій, котрий буде вести відлік та відображати реальний час в поточний момент. Такий пристрій має назву «годинник».

На даний момент на ринку існує величезна кількість годинників, з різною кількістю додаткового функціоналу, окрім мінімально необхідних ведення відліку поточного часу та його відображення — зокрема, з наявними функціями будильнику, календаря, таймера, автоматичного оновлення часу згідно літнього/зимового часу, згідно часового поясу тощо.

В рамках даної роботи розглядатиметься розробка аналогічного пристрою, з параметрами, що задані умовами технічного завдання, що дасть змогу розробити пристрій з точно витриманою кількістю додаткових функцій.

1. Опис структури пристрою та його складових

1.1. Структура пристрою.

Структурна схема пристрою наведена на наступному рисунку:

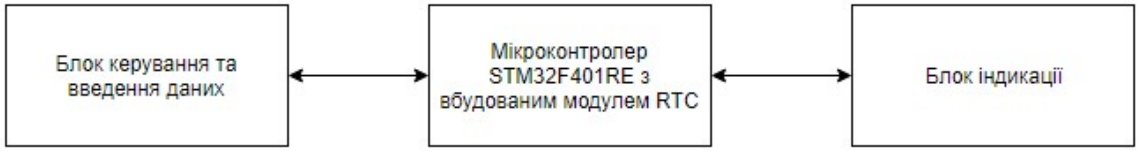

Рисунок 1.1 – Структурна схема пристрою.

Пристрій, що поставлено за мету розробити в рамках даної розрахунково-графічної роботи, являє собою пристрій відображення реального часу на основі STM32 з функціями календаря, будильника, світловою індикацією роботи пристрою та можливістю здійснення відповідних необхідних налаштувань доступних режимів роботи.

Основним компонентом даного пристрою є мікроконтролер STM32F401RE, який є наявним, зокрема, на налагоджувальній платі STM32F401 Nucleo.

Блок керування та введення даних представлений трьома тактовими кнопками з самоповерненням, за допомогою яких користувач може конфігурувати роботу пристрою — зокрема, налаштовувати поточне значення часу, активувати будильник та налаштовувати параметри його роботи, а також перемикати режими роботи пристрою.

Блок індикації реалізує візуальну індикацію роботи пристрою, і представлений світлодіодом та LCD-дисплеєм типу 1602 на основі контролера HD44780, а також розширювача портів вводу-виводу PCF8574.

1.2. Принципи та засоби введення даних.

Для конфігурації роботи пристрою використовуються три тактові кнопки з самоповерненням — SB2, SB3 та SB4, відповідно, на схемі електричній принциповій пристрою ДК61.403231.001Е3, підключені до пінів 13, 14 та 15 порту В мікроконтролера. Використовується внутрішня підтяжка даних пінів до лінії живлення мікроконтролера.

При натисканні на кнопку SB2, відбувається інкремент — збільшення на «1» значень відповідних полів дати та часу. При натисканні на кнопку SB3 — відбувається декремент — зменшення на «1» значень відповідних полів дати та часу. При цьому, відповідні зміни відразу відображаються на LCD-дисплеї.

Кнопка SB4 використовується для вибору режиму роботи пристрою. Для даної кнопки, за допомогою таймеру TIM2 та переривань, реалізовано визначення тривалості її натискання, від чого залежить те, який режим роботи пристрою буде вибрано. Загалом, даний принцип функціонує наступним чином: при натисканні на кнопку SB4 та утриманні її в натисненому положенні протягом проміжку часу, що більший за 7 секунд, відбувається деактивація будильника; при утриманні даної кнопки в натисненому положенні більше 5 секунд та менше 7 секунд — відбувається активація режиму налаштування поточних значень дати та часу; при утриманні даної кнопки в натисненому положенні протягом проміжку часу, що більший за 2 секунди та менший за 5 секунд — відбувається перехід в режим налаштування дати та часу будильника; при короткочасному утриманні даної кнопки — менше 2 секунд — якщо пристрій знаходився в режимі налаштування поточних дати та часу, або ж дати та часу будильника — відбуватиметься вибір наступного поля дати або часу для налаштування його значення за допомогою тактових кнопок SB2 та SB3; якщо пристрій

знаходився в стані, коли спрацював будильник — відбудеться вимкнення режиму спрацювання будильника, тобто, буде вимкнено відповідну індикацію спрацювання будильника; якщо пристрій знаходився в режимі відображення поточних дати та часу, або ж відображення встановлених дати та часу будильника — відбудеться перемикання між цими режимами.

Реалізований також програмний захист від «дріб'язку контактів» — так званий «дебаунс» («debounce») тактових кнопок. Значення тривалості «дебаунсу» задається в програмі, і може бути зміненим за бажанням. За замовчуванням воно рівне 250 мс — на цей проміжок часу після натискання будь-якої з кнопок зчитування вхідних значень з неї та інших кнопок зупиняється, шляхом блокування зовнішніх переривань, які можуть бути викликані натисканням даних тактових кнопок. Таким чином, це також не дає змогу користувачу виконувати введення даних частіше, ніж один раз в період «дебаунсу». Відлік інтервалу часу реалізації даної функції реалізований за допомогою системного таймеру — SysTick, а також відповідних переривань системного таймеру.

1.3. Принципи та засоби відображення даних та індикації роботи пристрою.

В якості дисплею було вибрано рідкокристалічний дисплей типу 1602 на основі контролеру HD44780, котрий містить два рядки символів з 16 знакомісцями в кожному.

Відображення даних, що передбачені вибраним режимом роботи пристрою, на дисплеї, відбувається постійно, при цьому оновлення даних, що відображаються, відбувається одразу після зміни користувачем їх значень шляхом натискання відповідних тактових кнопок.

Підключення дисплею до портів мікроконтролера відбувається за допомогою розширювача портів вводу-виводу PCF8574AT, зв'язок з яким відбувається за допомогою послідовного інтерфейсу передачі даних I2C, що дає змогу обмежитись всього двома, окрім ліній живлення та «землі» розширювача портів вводу-виводу, лініями — даних (SDA) та тактового сигналу (SCL) — для реалізації даного зв’язку. Детальніше інтерфейс I2C та принцип роботи з розширювачем портів вводу-виводу PCF8574AT в контексті реалізації поставленого завдання, будуть описані далі, в відповідних розділах.

Для роботи з LCD дисплеєм, підключеним до мікроконтролера за допомогою PCF8574AT, використовується написана власноруч бібліотека `lcd1602.h`, з використанням у її реалізації виключно функцій бібліотеки CMSIS, за допомогою функцій, наданих якою, в програмі керування пристроєм відбувається як початкова конфігурація роботи дисплея, так і подальша передача йому потрібних інструкцій керування або ж даних для відображення.

Також в якості елемента індикації роботи даного пристрою використовується світлодіод зеленого кольору. Даний світлодіод активується в режимі спрацювання будильника — таким чином, відбувається візуальна індикація сигналу будильника. Також в режимі спрацювання будильника відбувається виведення на дисплей текстового повідомлення, що сигналізує про відповідну подію, дублюючи, таким чином, описану вище світлодіодну індикацію.

2. Проектування схеми електричної принципової

Схема електрична принципова пристрою наведена у документі ДК61.403231.001Е3.

2.1. Вибір та обґрунтування елементної бази для реалізації блоків пристрою.

Розглянемо детальніше вибір елементної бази пристрою.

2.1.1. Вибір мікроконтролера.

Головним компонентом схеми, який забезпечує функціонування пристрою згідно передбаченого розробником алгоритму роботи, відображеного в вихідному коді — програмному забезпечені, є мікроконтролер. Для реалізації вимог, поставлених в технічному завданні, для використання в даному пристрої було вибрано потужний мікроконтролер STM32F401RE компанії STMicroelectronics, що є розміщеним на налагоджувальній платі STM32F401 Nucleo.

Використання налагоджувальної плати, загалом, значно спрощує роботу з мікроконтролером при створенні прототипу пристрою, так як на ній вже реалізована необхідна «обв'язка» МК у вигляді резисторів, конденсаторів, зовнішніх кварцових генераторів та інших необхідних для старту та стабільної роботи мікроконтролера компонентів.Також на даній платі наявний завантажувач-налагоджувач ST-Link, який значно спрощує процес відлагодження роботи програми.

Загалом, серед причин вибору саме даного компоненту, можна виділити наступні:

- висока продуктивність роботи при невисокому енергоспоживанні;

- популярність — мікроконтролери сімейства STM є дуже популярними в наш час, що забезпечує високу активність спільноти користувачів, і, як наслідок — велику кількість різного роду навчальної та довідкової інформації стосовно роботи з даними мікроконтролерами, швидке реагування виробника на відгуки та зауваження користувачів стосовно роботи його продукції;
- невисока ціна налагоджувальних плат з мікроконтролерами даного сімейства, котрі містять всі необхідні для запуску та стабільної роботи компоненти — так звану «обв'язку», а також програматори, котрі, окрім функції завантаження програмного забезпечення в пам'ять мікроконтролеру, дають також змогу здійснювати відлагодження його роботи в режимі реального часу, спостерігаючи за вмістом регістрів, станом периферії тощо;
- наявність у вільному доступі детальної документації, котру надає виробник;
- наявність дуже широкого набору периферії в складі даного МК — зокрема, що важливо — наявність вбудованого модуля RTC;
- особистий досвід розробки програмного забезпечення для мікроконтролерів STM32 серії F4 — що, власне, є дуже вагомим аргументом на користь вибору саме даного мікроконтролеру для використання в пристрої, що розробляється в рамках даної курсової роботи, так як дозволяє створити програмне забезпеченя в мінімальні терміни;

Отже, вибір саме мікроконтролеру STM32F401RE є цілком аргументованим та прийнятним в рамках поставленного завдання.

2.1.2. Компоненти блоку введення даних.

Блок введення даних на схемі електричній принциповій пристрою представлений тактовими кнопками з самоповерненням SB2, SB3, SB4. Як видно зі схеми, дані кнопки підтягнуті внутрішніми резисторами R11, R13 та R14, опором 40 кОм, до лінії +3,3В живлення мікроконтролера.

При складанні прототипу пристрою використовувались тактові кнопки TACTS-24N-F. Вони мають невеликі розміри а також низьку ціну, а, отже, є цілком прийнятними для використання в рамках реалізації поставленого завдання.

Позбавлення від «дріб'язку» контактів даних кнопок — програмне.

2.1.3. Компоненти блоку індикації.

Блок індикації представлений рідкокристалічним дисплеєм типу 1602 на основі контролера HD44780, а також розширювачем портів вводу-виводу на базі мікросхеми PCF8574AT, що використовується для його підключення до мікроконтролера STM32F401RE, і дає змогу керувати дисплеєм з використанням послідовного інтерфейсу передачі даних I2C — тобто, використовуючи лише лінію даних SDA та лінію тактового сигналу SCL, а також лінії живлення та «землі» для живлення даних схем.

Мікросхема PCF8574AT, згідно документації, передбачає частоту тактового сигналу на лінії SCL, рівну 100 кГц. Адреса мікросхеми, в контексті комунікації по інтерфейсу I2C — 0x3F.

Зовнішні підтягуючи резистори — резистори R3 та R4, опором 4.7 кОм, «підтягують» лінії даних та тактового сигналу — SDA та SCL — інтерфейсу I2C, до лінії живлення +5В, що передбачено принципом функціонування послідовного інтерфейсу I2C.

При складанні прототипу в якості даних резисторів було використано виводні (монтаж «крізь отвір» — Through hole (TH)) резистори МЛТ-0,5 та

МЛТ-0,125, потужністю 0,5 та 0,125 Вт відповідно. В контексті даного пристрою, до цих резисторів не ставляться особливі вимоги, так як вони не знаходяться в гілках схеми, котрі характерні протіканням в них струмів високого значення, і були вибрані, при створенні прототипу, виходячи з резисторів, що були у наявності, а також виходячи з загальноприйнятих конструкторських практик, тому, в рамках даної роботи, за рахунок невисокої складності схеми, а також малої кількості пристроїв на лініях I2C, такий вибір є цілком прийнятним.

Залишилось розглянути вибір світлодіоду зеленого кольору HL1, що використовується в даному пристрої в складі блоку індикації. Загалом, на ринку наявна велика кількість світлодіодів, що задовольняють поставлені вимоги, і мають невисоку ціну, малі габаритні розміри тощо. В даному випадку, зупинимось на світлодіоді 19-213SY6C/S530-E2/TR8LED, котрий виробляється компанією Everlight. Даний світлодіод наявний в корпусі SMD-типу, що дає змогу застосувати поверхневий монтаж. Також він володіє прийнятними світловими характеристиками, та малими габаритними розмірами. Крім цього, такий світлодіод наявний на налагоджувальній платі STM32F401 Nucleo.

2.2. Необхідні схемотехнічні розрахунки.

Здійснимо розрахунок опору струмообмежуючого резистору R1 для світлодіоду HL1.

Для цього звернемося до документації на даний світлодіод, інформація звідки наведена на рисунку

Electro-Optical Characteristics (Ta=25℃)

Parameter	Symbol	Min.	Typ.	Max.	Unit	Condition
Luminous Intensity	I_v	16	21	-----	mcd	I_F=20 mA
Viewing Angle	2θ 1/2	-----	100	-----	deg	
Peak Wavelength	λ_p	-----	575	-----	nm	
Dominant Wavelength	λ_d	-----	573	-----	nm	
Spectrum Radiation Bandwidth	$\triangle\lambda$	-----	20	-----	nm	
Forward Voltage	V_F	1.7	2.0	2.4	V	
Reverse Current	I_R	-----	-----	10	μA	V_R=5V

Рисунок 2.1 — Електро-оптичні характеристики світлодіоду 19-213SY6C/S530-E2/TR8LED, взяті з документації.

Як видно з Рисунку 2.1, типове значення прямого падіння напруги на даному світлодіоді при протіканні крізь нього прямого струму, рівного 20 мА, становить: $V_F = 2$ В.

Зі схеми електричної принципової пристрою ДК61.403231.001Е3 видно, що даний світлодіод підключений до піну 5 порту А мікроконтролера STM32F401RE. Вихідна напруга при високому логічному рівні на виході пінів даного мікроконтролера рівна: $V_{out} = 3.3$ В.

В якості значення струму, що буде протікати через світлодіод, виберемо $I_F = 20$ мА. Максимальне значення прямого струму через даний світлодіод, згідно документації — 25 мА. Відповідно, обране значення струму є меншим за максимальне, і тому — є прийнятним.

Таким чином, маємо:

$$R_1 = \frac{V_{out} - V_F}{I_F} = \frac{3.3 - 2}{20 \cdot 10^{-3}} = 65\ (\text{Ом})$$

Отже, для забезпечення прямого струму, рівного 20 мА, через даний світлодіод, при підключенні його до піна порту мікроконтролера, вихідна

напруга при високому рівні на якому становить 3.3 В, за умови, що пряме падіння напруги на даному світлодіоді становить 2 В, при протіканні крізь нього струму, рівного 20 мА, необхідно використати резистор опором 65 Ом.

При складанні прототипу, при цьому, в якості резистору R1 було використано резистор опором 510 Ом, наявний на налагоджувальній платі STM32F401 Nucleo, що забезпечує, таким чином, прямий струм через світлодіод, що рівний приблизно 2-3 мА, чого, втім, цілком достатньо для здійснення світлової індикації в рамках пристрою, що розробляється.

3. Розробка програми керування пристроєм

Вихідний код програми керування пристроєм наведено в Додатку Б.

Програма керування пристроєм реалізована виключно з використанням бібліотеки CMSIS, з реалізацією, таким чином, прямого та послідовного доступу до конфігураційних регістрів керування периферією мікроконтролера. Такий підхід забезпечує максимально точну, після мови ассемблеру, а також повністю контрольовану та усвідомлювану розробником, роботу з периферією. Крім цього, важливим аргументом на користь використання бібліотеки CMSIS є отримання, при її використанні, максимально оптимального та компактного вихідного коду програми.

3.1. Призначення регістрів та змінних, а також використаних портів мікроконтролера.

Нижче наведено перелік використаних в програмі керування пристроєм регістрів, змінних та портів мікроконтролера, а також їх короткий опис та призначення.

Використані виводи мікроконтролера:

- PB13 — Підключення тактової кнопки для здійснення інкрементації відповідних значень дати та часу при здійсненні налаштувань;
- PB14 — Підключення тактової кнопки для здійснення декрементації відповідних значень дати та часу при здійсненні налаштувань;
- PB15 — Підключення тактової кнопки для здійснення вибору режиму функціонування пристрою;
- PB8 — Вихід лінії тактового сигналу (SCL) інтерфейсу I2C1, підключення зовнішнього підтягуючого резистору;
- PB9 — Вихід лінії даних інтерфейсу (SDA) I2C1, підключення зовнішнього підтягуючого резистору;

- +5V — Підключення входу живлення LCD дисплею, підключення зовнішніх резисторів для підтяжки ліній SCL та SDA до напруги живлення;
- GND — Підлючення «землі» до контактів тактових кнопок та відповідного входу LCD дисплею;

Використані в програмі керування пристроєм регістри мікроконтролера:

- SysTick->CTRL — Регістр керування конфігурацією системного таймера;
- SysTick->LOAD — Значення, яке буде використане при перезавантаженні системного таймеру після його обнулення
- SysTick->VAL — Поточне значення системного таймера (регістр лічби/рахунку);
- EXTI->IMR — Регістр маскування зовнішніх переривань;
- RCC->APB1ENR — Регістр конфігурації тактування шини APB1, використовується для увімкнення тактування таймеру TIM2, а також тактування модуля I2C1;
- TIM2->PSC — Налаштування значення переддільника таймера TIM2;
- TIM2->ARR — Регістр автоперезавантаження таймера TIM2;
- TIM2->DIER — Регістр налаштування дозволів переривань таймера TIM2 (DMA/Interrupt enable register);
- TIM2->CR1 — Регістр контролю (control register) таймера TIM2, використовується для вибору режиму рахунку «вгору»;
- TIM2->SR — Регістр статусу таймера TIM2, використовується для маніпуляцій з так званими «pending-бітами» в обробнику відповідних переривань;
- GPIOB->MODER — Конфігураційний регістр порту вводу –виводу B, використовується для вибору режиму роботи на вхід, а також для

вибору режиму альтернативної функції для відповідних пінів порту B;

- GPIOB->PUPDR — Конфігураційний регістр порту вводу –виводу B, використовується для увімнення внутрішньої підтяжки до живлення пінів PB13, PB14 та PB15, а також для конфігурації пінів, які використовуються модулем I2C1;
- RCC->APB2ENR — Регістр конфігурації тактування шини APB2, використовується для увімкнення тактування контролеру системної конфігурації (SYSCFG, System Configuration Controller), що здійснює керування системою зовнішніх переривань;
- SYSCFG->EXTICR — Регістр для налаштування вибору джерел зовнішніх переривань;
- EXTI->IMR — Регістр для налаштування маскування запитів на переривання від зовнішніх джерел;
- EXTI->FTSR — Регістр для налаштування вибору у якості «тригеру», який викличе переривання від зовнішнього джерела, появу заднього фронту вхідного сигналу на вході відповідного джерела зовнішніх переривань;
- EXTI->PR — Регістр, що містить біти, які сигналізують про появу запиту на переривання на якійсь із ліній EXTI;
- RCC->AHB1ENR — Регістр для конфігурації тактування шини AHB1, використовується для увімкнення тактування порту вводу-виводу GPIOB;
- GPIOB->AFR — Регістр налаштування вибору альтернативної функції пінів портів вводу-виводу, використовується для налаштування роботи пінів PB8 та PB9 в якості виходів ліній тактування та даних модуля I2C1;

- GPIOB->OTYPER — Конфігураційний регістр портів вводу-виводу, використовується для налаштування режиму роботи пінів PB8 та PB9 в режимі «open drain»;
- I2C1->CR2 — Конфігураційний регістр модуля I2C1 , використовується для задання частоти тактування шини PB1, на якій розміщений даний модуль;
- I2C1->CCR — Конфігураційний регістр модуля I2C1, використовується для задання значення частоти вихідного сигналу тактування (Serial Clock — SCL), який буде генеруватись модулем I2C1, а також для вибору режиму роботи Fast/Standard;
- I2C1->TRISE — Конфігураційний регістр модуля I2C1, використовується для задання параметрів «таймінгів» сигналу SCL, який буде генеруватись відповідним модулем — значень тривалості фронтів, тощо;
- I2C1->CR1 — Конфігураційний регістр модуля I2C1, використовується для увімкнення та зупинки роботи модулю I2C1, а також для ініціалізації старт та стоп-бітів;
- I2C1->DR — Регістр даних модулю I2C1, куди поміщаються дані, які мають бути відправлені по лінії даних;
- I2C1->SR1 — Регістр статусу модуля I2C1, використовується для слідкування за процесом передачі/прийому даних відповідним модулем, шляхом перевірки значень відповідних бітових полів в даному регістрі, які змінюютьс свої значення відповідно до подій, які відбуваються при передачі/прийомі даних модулем — полів ініціалізації старт-біту, успішного надсилання адреси Slave-пристрою, сигналу про наявність вільного місця в регістрі даних, а також сигналу про закінчення передачі;

- I2C1->SR2 — Регістр статусу модуля I2C1, використовується для слідкування за процесом передачі/прийому даних відповідним модулем, шляхом перевірки значень відповідних бітових полів в даному регістрі, які змінюютьс свої значення відповідно до подій, які відбуваються при передачі/прийомі даних модулем;
- 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, з його допомогою можна ввести годинник в однойменний режим;
- 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, 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;

Використані в програмі керування пристроєм глобальні змінні та їх призначення і опис:

- uint8_t LCD_show_ready — Свідчить про зміну значень вхідних даних і потребу оновити виведені на дисплей значення;
- uint8_t debounce_ms_enable — «Прапорець», який свідчить про запуск роботи алгоритму ліквідації «дріб'язку контактів»;

- uint8_t pressure_flag — «Прапорець», який зберігає в собі значення, яке містилось в регістрі IDR для відповідного піна, до якого підключена тактова кнопка, для ідентифікації події тривалого натискання на дану кнопку;
- uint16_t pressure_10_ms_ticks — Лічильник, який збільшується кожні 10 мс в відповідній функції, для ідентифікації тривалого натискання на вибрану тактову кнопку;
- uint32_t field — Змінна, що використовується для послідовного «проходу» по полях дати та часу, та містить номер поточного поля, значення якого конфігурується в поточний момент часу;
- uint32_t alarm_fields_num — Кількість полів дати та часу, в яких міститься інформація про те, коли відбудеться спрацювання будильника;
- uint32_t clock_fields_num — Кількість полів дати та часу, що містять інформацію про поточну дату та час;
- uint32_t alarm_cfg_mode — Значення змінної рівне одиниці, коли активований режим налаштування дати та часу спрацювання будильника;
- uint32_t clock_cfg_mode — Значення змінної рівне одиниці, коли активований режим налаштування поточних дати та часу;
- uint32_t alarm_show_mode — Значення змінної рівне одиниці, коли активований режим відображення поточних налаштувань дати та часу спрацювання будильника на екрані;
- uint32_t clock_show_mode — Значення змінної рівне одиниці, коли активований режим відображення на екрані поточних значень дати та часу;

- uint32_t btn_pressure_start — Значення змінної встановлюється в «1», коли відбувається натискання на відповідну тактову кнопку, і починається відлік часу натискання;
- uint32_t btn_pressure_done — Значення змінної встановлюється в «1», коли закінчується натискання на відповідну тактову кнопку, і відлік часу натискання закінчився;
- uint8_t time_get_done — Значення змінної рівне одиниці за умови успішного виконання процедури отримання поточних значень дати та часу;
- uint8_t alarm_get_done — Значення змінної рівне одиниці за умови успішного виконання процедури отримання поточних значень дати та часу спрацювання сигналу будильника;
- uint8_t clk_1hz — Значення змінної встановлюється в «1» з частотою 1 Гц — необхідна для реалізації періодичного оновлення даних на дисплеї;
- uint8_t alarm_enable — Значення змінної рівне одиниці, коли активований режим спрацювання сигналу будильника;
- uint8_t is_alarm — Значення змінної рівне одиниці в момент спрацювання сигналу будильника;
- RTC_struct_full RTC_data_full_buff — Структура, що використовується для зберігання повної інформації про дату та час: значень року, дня тижня, місяця, дня місяця, годин, хвилин та секунд — у форматі «десятки — окремо, одиниці — окремо», що є зручним при роботі зі значеннями дати та часу, котрі представлені в BCD форматі у відповідних регістрах модуля RTC;
- RTC_struct_brief RTC_data_brief_buff — Структура, що використовується для зберігання стислої інформації про дату та час: значень року, дня тижня, місяця, дня місяця, годин, хвилин та

секунд — у звичайному форматі, без розділення на десятки та одиниці, що зручно при введенні відповідних значень за допомогою тактових кнопок;

Типи даних — для використання необхідно підключити бібліотеку <stdint.h>:

- uint8_t — тип даних, що являє собою беззнакове ціле восьмирозрядне число;
- uint16_t — тип даних, що являє собою беззнакове ціле шістнадцятирозрядне число;
- uint32_t — тип даних, що являє собою беззнакове ціле тридцятидвохрозрядне число;

3.2. Перелік та короткий опис використаних в програмі керування пристроєм функцій.

- Опис функції init_systick:

Дана функція використовується для стартової ініціалізації роботи системного таймеру, задаючи початкове значення в регістрі перезавантаження, вмикаючи тактування від системної частоти, вмикаючи дозвіл переривань, а також вимикаючи системний таймер.

- Опис функції SysTick_Handler:

Обробник переривань системного таймеру. Використовується в алгоритмі ліквідації «дріб'язку контактів» тактових кнопок. В даному обробнику відбувається дозвіл замаскованих раніше, в обробнику переривань EXTI, переривань, що надходять від зовнішніх джерел, а також обнулення «прапорця», що сигналізує про активну роботу алгоритму «дебаунсу», і вимкнення після цього роботи системного таймеру.

- Опис функції systick_debounce_ms:

Дана функція приймає в якості аргументу значення часу в мілісекундах, протягом якого відбуватиметься «дебаунс» тактових кнопок, і заносить його в

потрібному вигляді, після деяких арифметичних маніпуляцій згідно документації, до регістру перезавантаження системного таймеру, виставляючи при цьому в активне значення «прапорець», що сигналізує про активну роботу алгоритму «дебаунсу», і вмикаючи після цього роботу системного таймеру.

- Опис функції tim2_init:

Функція, що використовується для початкової ініціалізації роботи таймеру TIM2.

- Опис функції TIM2_IRQHandler:

Обробник переривань таймеру TIM2.

- Опис функції btn_pressure_check:

Перевірка тривалого натискання на тактову кнопку.

- Опис функції init_GPIO:

Конфігурація роботи потрібних для реалізації поставленого завдання портів вводу-виводу.

- Опис функції btn_irq_init:

Налаштування переривань від зовнішніх джерел.

- Опис функції EXTI15_10_IRQHandler:

Обробник переривань від зовнішніх джерел.

- Опис функції softdelay:

Реалізація часової затримки за допомогою системи циклів.

- Опис функції I2C1_init:

Ініціалізація модулю I2C1.

- Опис функції I2C_send:

Функція, що виконує надсилання даних по інтерфейсу I2C.

- Опис функції PCF8574AT_send:

Перетворення команд керування контролером LCD дисплею, переданих у функцію, у формат інструкцій, з яким працює PCF8574AT, а також відправка

сформованих інструкцій до PCF8547AT за допомогою інтерфейсу I2C та функції I2C_send.

- Опис функції LCD1602_backlight_on:

  Функція, яка вмикає підсвічування дисплею.

- Опис функції LCD1602_backlight_off:

  Функція, яка вимикає підсвічування дисплею.

- Опис функції LCD1602_send_char:

Функція для виведення символу в ASCII-форматі, згідно таблиці передбачених знакогенератором та збережених в CGRAM пам'яті контролеру дисплея, символів, на дисплей, без вибору адреси знакомісця для виведення.

- Опис функції LCD1602_init:

  Функція, що виконує ініціалізацію LCD дисплея.

- Опис функції LCD1602_set_cursor:

Запис бажаної адреси в вказівник DDRAM — встановлення позиції курсора:

- Опис функції LCD1602_send_char_position:

Функція для виведення символу в ASCII-форматі, згідно таблиці передбачених знакогенератором та збережених в CGRAM пам'яті контролеру дисплея, символів, на дисплей, з вибором адреси знакомісця для виведення.

- Опис функції LCD1602_position_rst:

Скидання значення вказівника адреси DDRAM в значення 0x0 — на початок діапазону адрес.

- Опис функції LCD1602_display_clear:

  Очищення дисплею від виведених символів.

- Опис функції LCD1602_send_string:

Функція для для виведення рядка тексту на екран, з можливістю попереднього встановлення адреси DDRAM — позиції початку виведення.

- Опис функції LCD1602_send_integer:

Виведення числа в діапазоні від 0 до 99999 на дисплей, з можливістю попереднього встановлення адреси DDRAM — позиції початку виведення.

- Опис функції fill_struct_default:
  Заповнення полів екземплярів структур RTC_struct_brief та RTC_struct_full значеннями за замовчуванням.
- Опис функції Nucleo_LED_init:
  Конфігурування роботи піну мікроконтролера, до якого приєднаний світлодіод на налагоджувальній платі STM32F401 Nucleo, що використовується для візуальної індикації сигналу спрацювання будильника.
- Опис функції btn_fill_date_fields:
  Функція для послідовного «проходу» полями структури типу RTC_struct_brief та їх заповнення відповідними значеннями дати та часу, що вводяться користувачем за допомогою пристроїв введення.
- Опис функції RTC_auto_wakeup_enable:
  Налаштування wakeup-режиму роботи модуля RTC.
- Опис функції RTC_WKUP_IRQHandler:
  Обробник переривань, що генеруються wakeup-режимом роботи модуля RTC.
- Опис функції RTC_data_init:
  Початкове налаштування регістрів дати та часу модуля RTC та формату інтерпретації відповідних значень в даних регістрах.
- Опис функції RTC_data_update:
  Налаштування поточних значень дати та часу модуля RTC.
- Опис функції RTC_alarm_init:
  Ініціалізація та початкова конфігурація режиму будильника модуля RTC.

- Опис функції RTC_alarm_update:
  Активація сигналу будильника модуля RTC, активація генерації відповідних переривань.
- Опис функції RTC_alarm_disable:
  Деактивація сигналу будильника модуля RTC та очистка відповідних значень дати та часу спрацювання будильника в відповідному регістрі модуля RTC.
- Опис функції RTC_Alarm_IRQHandler:
  Обробник переривань, що генеруються при спрацюванні сигналу будильника модуля RTC.
- Опис функції fill_RTC_struct_full:
  Заповнення полів структури типу RTC_struct_full значеннями, сформованими на основі значень полів структури типу RTC_struct_brief — конвертування значень дати та часу зі «стислого» формату в формат «десятки — окремо, одиниці — окремо».
- Опис функції RTC_get_time:
  Внесення в структуру типу RTC_struct_brief поточних значень дати та часу.
- Опис функції RTC_get_alarm:
  Внесення в структуру типу RTC_struct_brief поточних встановлених значень дати та часу спрацювання будильника.
- Опис функції LCD_clock_display:
  Виведення на дисплей необхідної інформації про поточну дату та час, або ж конфігураційної інформації, згідно вибраного режиму функціонування пристрою.
- Опис функції RTC_init:
  Ініціалізація роботи модуля RTC.
- Опис функції LED_indication:

Реалізація візуальної індикації спрацювання сигналу будильника за допомогою світлодіоду.

- Опис функції main:

Головна функція програми. В ній виконується ініціалізація всієї необхідної для роботи периферії, і перехід після цього в нескінченний цикл while(1).

3.3. Налаштування системи тактування мікроконтролера.

Схему налаштування системи тактування мікроконтролера можна бачити на Рисунку 3.1.

Для генерації вихідного коду, який здійснюватиме конфігурацію роботи мікроконтролера згідно заданих параметрів, було використано спеціальний інструмент, який доступний для завантаження на веб-сайті STMicroelectronics.

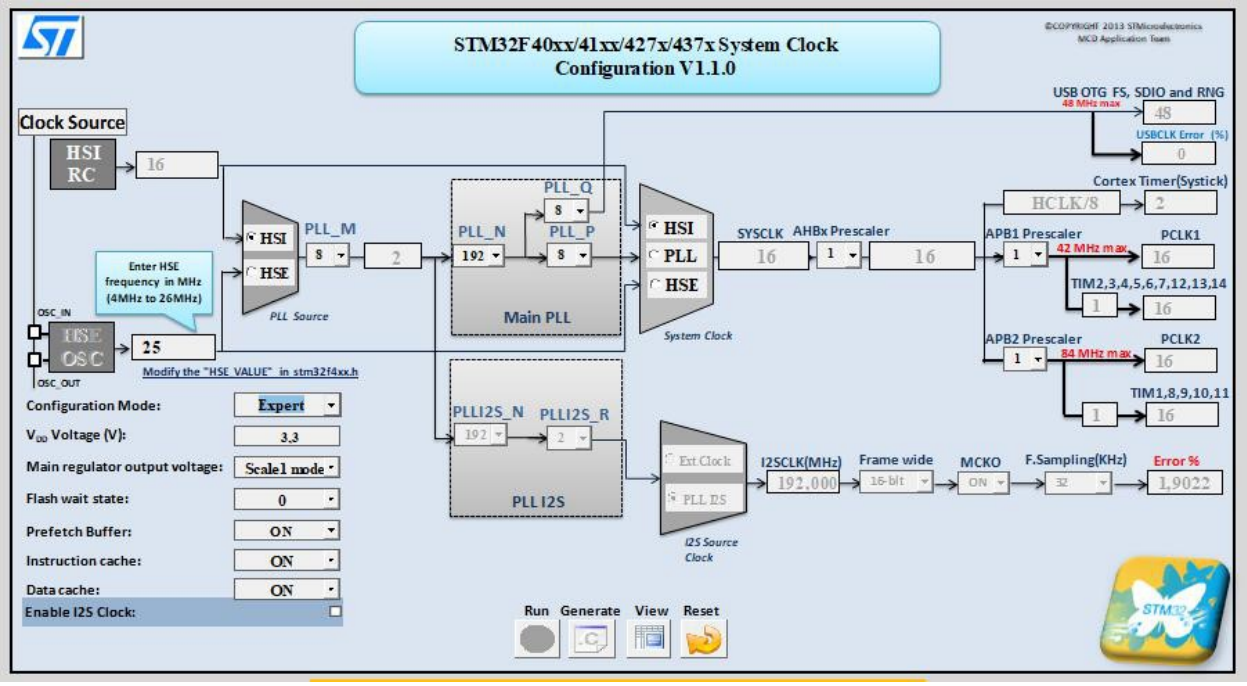

Рисунок 3.1 — Конфігурація системи тактування мікроконтролера.

Як можна бачити з рисунку, дана конфігурація передбачає роботу мікроконтролера з системною частотою (SYSCLK), рівною 16 МГц. Для

цього використовується внутрішній високошвидкісний RC-генератор (HSI — High Speed Internal), який і забезпечує генерацію тактового сигналу з даною частотою. Значення коефіцієнтів переддільників для шин, до яких приєднана периферія, рівні 1, відповідно, системна частота не буде зменшуватись, і шини APB та AHB будуть тактуватись від частоти 16 МГц. Ця ж частота (або ж поділена на 8, залежно від налаштувань) буде тактувати і системний таймер SysTick.

3.4. Опис алгоритму та принципу роботи з дисплеєм.

3.4.1. Опис послідовного інтерфейсу передачі даних I2C.

В інтерфейсі використовуються дві лінії — це лінія тактування SCL, і лінія передачі даних SDA, які разом утворюють шину даних. Пристрої, підключені до шини діляться на ведучого і веденого. Ведучий ініціалізує процес передачі даних і видає тактові імпульси на лінію SCL, ведений приймає команди/дані, а також видає дані за запитом ведучого. Лінії SDA і SCL двонаправлені, пристрої, що підключаються до шини, повинні мати виводи, переналаштовуємі на вхід та вихід. Причому тип виходу має бути з відкритим коллектором/стоком, в зв'язку з чим, обидві лінії SDA і SCL через резистори підтягуються до позитивного полюса джерела живлення. На наступному рисунку приведена схема підключення інтерфейсу I2C.

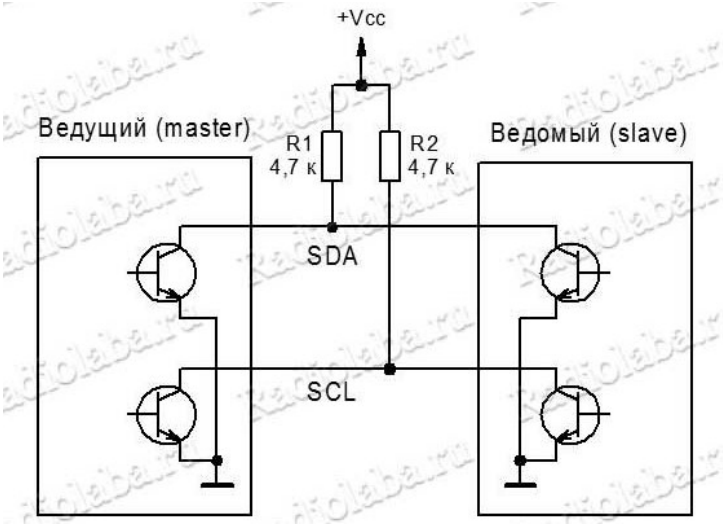

Рисунок 3.2 — Схема підключення пристроїв при використанні інтерфейсу I2C.

Інтерфейсом передбачена програмна адресація пристроїв підключених до шини, найбільш поширеною є довжина адреси в 7 біт, теоретично це дозволяє підключати на шину до 127 пристроїв, але частина адрес за специфікацією зарезервовані і не можуть використовуватися розробниками. Кожен пристрій має свою унікальну адресу, який закладений виробником, і який вказується в технічній документації. Адреса пристрою може бути фіксованою, або ж з можливістю апаратного налаштування — в цьому випадку пристрій має додаткові входи, і в залежності від рівня напруги на входах (високий або низький), можна отримати різні значення адреси. Зазвичай кількість таких входів варіюється від 1-го до 3-х, які задають значення певних бітів 7-бітної адреси. Апаратне налаштування адреси передбачене для можливості підключення декількох однотипних пристроїв на одну шину.

Кожен сеанс передачі даних починається так званим «старт-бітом». У початковому стані, коли шина вільна, обидві лінії SDA і SCL підтягнуті до високого логічного рівня. «Старт-біт» являє собою перемикання лінії SDA з високого логічного рівня на низький, в той час, коли на лінії SCL встановлений високий рівень.

Аналогічно, сеанс передачі даних завершується «стоп-бітом» — перемиканням лінії SDA з низького логічного рівня на високий, при високому рівні на лінії SCL. Дані умови генерує пристрій-ведучий (Master).

Виходячи з подій «старт» та «стоп», під час передачі даних лінія SDA може перемикатися тільки при низькому рівні на лінії SCL, тобто встановлення нових даних на лінії SDA можливе тільки після спаду логічного рівня на SCL. Протягом імпульсу тактування (високий рівень на SCL), стан лінії SDA не повинен змінюватися, так як в цей час виконується зчитування даних з лінії SDA.

Дані по інтерфейсу передаються побайтно, старшим бітом вперед, за кожним переданим байтом (8 біт) слідує біт підтвердження, пристрій (ведучий або ведений), що прийняв байт даних, встановлює низький рівень на лінії SDA на наступному тактовому імпульсі SCL, тим самим підтверджуючи отримання байту. В цей час передаючий пристрій має опитувати лінію SDA, чекаючи відповідь про успішне отримання байта. На наступному рисунку представлена діаграма передачі даних по шині I2C.

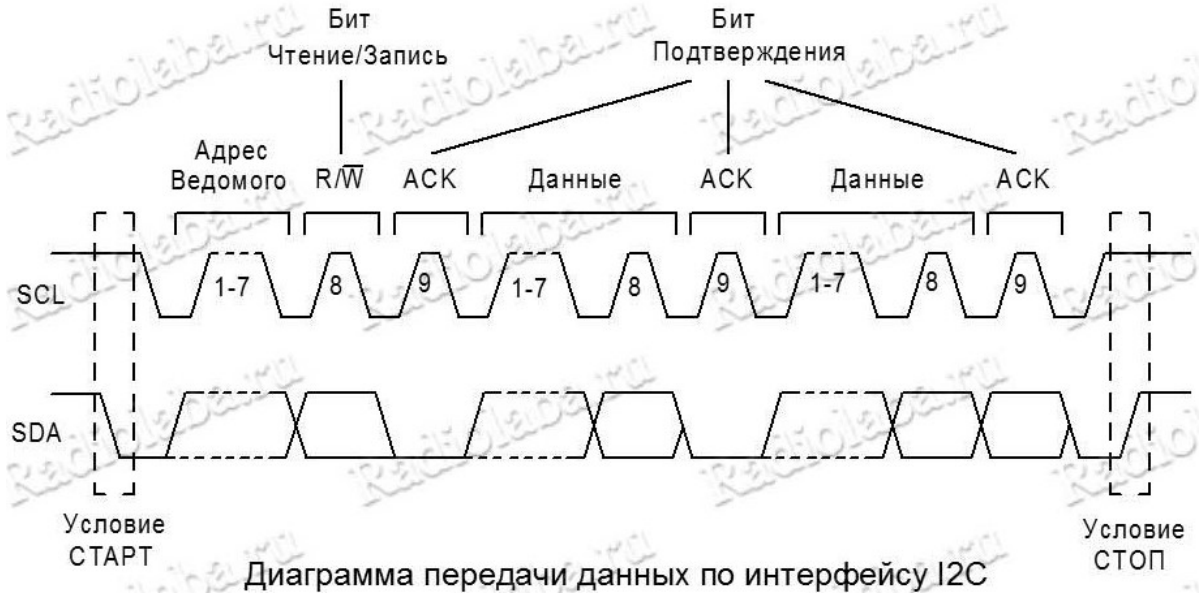

Диаграмма передачи данных по интерфейсу I2C

Рисунок 3.3 — Діаграма передачі даних по інтерфейсу I2C.

Спочатку передається байт з 7-бітовою адресою веденого, значення 8-го біта (R/!W) визначає напрямок передачі даних, нульове значення відповідає запису даних, тобто передача від ведучого до веденого. Якщо біт напрямку дорівнює 1, то виконується читання даних з веденого.

Ведений порівнює передану адресу зі своєю, і при збігу відгукується, встановлюючи низький рівень на лінії SDA (біт підтвердження). Ведучий, отримавши підтвердження, починає передавати байти даних, або ж приймає їх, залежно від вибраного напрямку передачі.

На наступному рисунку відображена діаграма передачі даних за допомогою інтерфейсу I2C при запису байту даних в визначений регістр від пристрою-ведучого до пристрою-веденого.

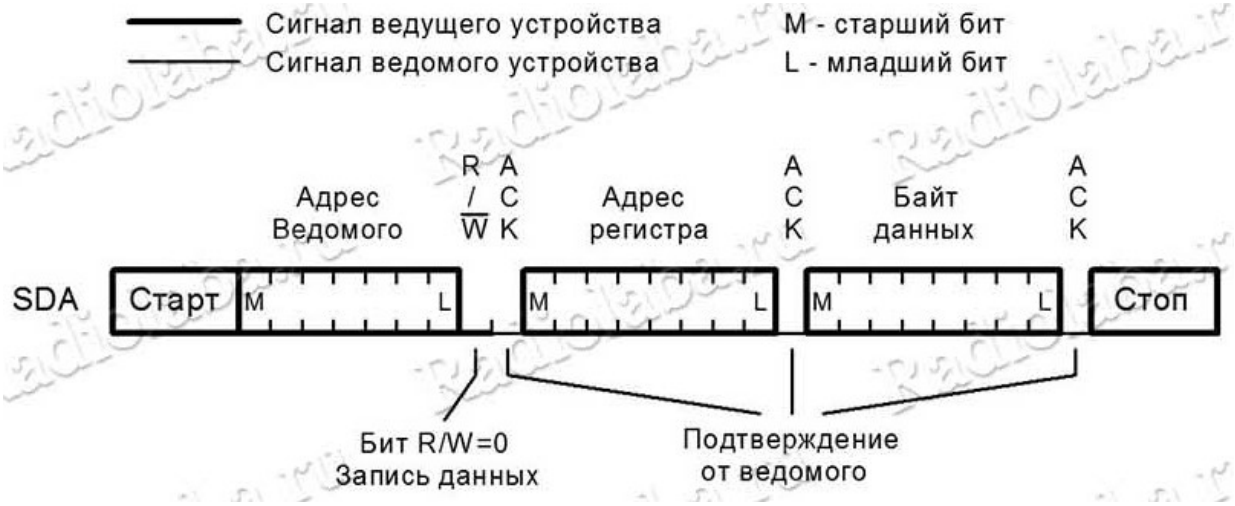

Рисунок 3.4 — Запис байту даних в заданий регістр.

3.4.2. Опис рідкокристалічного дисплею типу 1602 на основі контролера HD44780.

Даний дисплей містить 2 рядки по 16 символів. Напруга живлення може знаходитись в межах від 4.5 до 5.5В, струм споживання становить 1.2мА, без врахування підсвічування.

Дисплей має 16 виводів, їх призначення та опис наведені в наступній таблиці:

Таблиця 3.4 — Опис виводів LCD дисплея 1602 на базі контролера HD44780.

Номер виводу	Назва	Опис
1	Vss	Вивід живлення дисплея «-»
2	Vdd	Вивід живлення дисплея «+»
3	Vo	Вхід регулювання контрастності дисплея
4	RS	Вхід вибору типу інструкції: 1 — дані, 0 — команда
5	R/!W	Вхід напряму передачі даних: 1— запис даних в дисплей, 0 — читання даних з дисплею
6	E	Вхід тактування

7-14	DB0-DB7	Лінії вводу/виводу даних
15	A	Вивід живлення підсвічування «+»
16	K	Вивід живлення підсвічування «-»

Вхід VO призначений для регулювання контрастності екрану, яка залежить від величини напруги на вході, зазвичай для цих цілей встановлюється змінний резистор опором 10-20 кОм, підключений до лінії живлення. За допомогою входів RS, R/!W вибирається тип інструкцій та напрямок передачі даних. Вхід тактування E призначений для "замикання" (фіксації) станів входів і ліній введення/виведення, введення інструкцій в контролер дисплея — зчитування даних відбувається по спаду сигналу (по задньому фронту). Лінії DB7-DB0 представляють собою 8-бітний інтерфейс вводу/виводу даних, при цьому за один період тактового імпульсу передається 1 байт даних. Виводи A і K є анодом та катодом світлодіодів підсвічування, на платі дисплея встановлений струмообмежуючий резистор.

Команди для керування дисплеєм представлені на наступному рисунку:

№	Описание инструкции	Код инструкции										Время выполнения
		RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	Очистка дисплея с установкой курсора в начало первой строки	0	0	0	0	0	0	0	0	0	1	1,53 мс
2	Установка курсора в начало первой строки	0	0	0	0	0	0	0	0	1	–	1,53 мс
3	Установка направления вывода символов, разрешение сдвига экрана	0	0	0	0	0	0	0	1	I/D	SH	39 мкс
4	Управление режимом питания дисплея и отображением курсора	0	0	0	0	0	0	1	D	C	B	39 мкс
5	Команда сдвига курсора и экрана	0	0	0	0	0	1	S/C	R/L	–	–	39 мкс
6	Настройка интерфейса ввода/вывода данных, количества строк для вывода символов, размера шрифта	0	0	0	0	1	DL	N	F	–	–	39 мкс
7	Запись адреса CGRAM памяти в адресный указатель	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
8	Запись адреса DDRAM памяти в адресный указатель	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
9	Команда чтения флага занятости и адресного указателя	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	0 мкс
10	Запись данных во внутреннюю память дисплея	1	0	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс
11	Чтение данных из внутренней памяти дисплея	1	1	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс

Рисунок 3.5 – Таблиця, в якій представлені команди керування LCD дисплеєм типу 1602.

Призначення відповідних бітів пояснюється на Рисунку 3.2.

Бит	Значение	Описание
I/D	0	Вывод символов справа-налево, декремент адресного указателя DDRAM/CGRAM памяти
	1	Вывод символов слева-направо, инкремент адресного указателя DDRAM/CGRAM памяти
SH	0	Запрет сдвига экрана при выводе символов
	1	Разрешение сдвига экрана при выводе символов
D	0	Выключить экран дисплея, сегменты погашены, содержимое внутренней памяти сохраняется
	1	Включить экран дисплея, нормальный режим работы
C	0	Отключить отображение курсора
	1	Включить отображение курсора
B	0	Отключить функцию мигания курсора
	1	Включить функцию мигания курсора
S/C	0	Выбрать курсор для сдвига
	1	Выбрать экран (вместе с курсором) для сдвига
R/L	0	Сдвиг влево (только курсор или весь экран, зависит от бита S/C)
	1	Сдвиг вправо (только курсор или весь экран, зависит от бита S/C)
DL	0	4-битный интерфейс ввода/вывода данных
	1	8-битный интерфейс ввода/вывода данных
N	0	Использовать одну строку для вывода символов
	1	Задействовать 2 строки для вывода символов
F	0	Размер шрифта 5×8 пикселей
	1	Размер шрифта 5×11 пикселей
BF	0	Контроллер дисплея готов к обработке новой команды
	1	Контроллер дисплея занят выполнением внутренних операций

Рисунок 3.6 – Призначення бітів в інструкціях керування дисплеєм типу 1602.

Дисплей можна підключити до мікроконтролеру, використовуючи всі лінії вводу/виводу, тобто через 8-бітний інтерфейс, але для цього потрібна значна кількість виводів мікроконтролера. Можна скоротити кількість висновків, якщо переключитися на 4-бітний інтерфейс (використавши інструкцію №6 з таблиці на Рисунку 3.4.), в цьому режимі задіяними є тільки лінії вводу/виводу DB7-DB4, а лінії DB3-DB0 стають неактивними. У цьому випадку 1 байт даних передається за два тактових імпульси — спочатку старші 4 біти, потім молодші.

Внутрішня пам'ять LCD 1602 поділяється на 3 види: DDRAM, CGROM і CGRAM. Область DDRAM (Display Data RAM) пам'яті використовується для зберігання 8-бітного коду ASCII символів, що відображаються на екрані.

Адреси регістрів DDRAM пам'яті пов'язані з положенням символів на екрані, дана відповідність приведена на наступному рисунку:

	Адреса ячеек															
1-я строка	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2-я строка	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Рисунок 3.7 – Відповідність адрес пам’яті DDRAM положенню символів на екрані.

В кожному рядку екрану вміщається 16 символів, але це тільки видима область, загальний же обсяг DDRAM пам'яті становить 80 байт, тобто в кожен рядок можна записати по 40 символів, і тільки 16 з них будуть відображатися на екрані, інші символи при цьому залишаться в невидимій області.

За допомогою команди зсуву екрану (інструкція №5 в таблиці на Рисунку 3.4), можна переглянути всі інші символи. Регістри з адресами 0x00-0x27 складають перший рядок, комірки 0x40-0x67 — другий рядок. Якщо задати тільки один рядок для виведення (біт N = 0 в інструкції №6 на Рисунку 3.4), то довжина рядка збільшиться, і вміщатиме 80 символів.

Пам'ять CGROM (Character Generator ROM) являє собою знакогенератор і містить дані для відображення ASCII символів. У пам'яті закладені спецзнаки, цифри, латинський алфавіт. Кожен символ займає 5 байт в пам'яті, що відповідає розміру шрифту 5 × 8 пікселів. На наступному рисунку представлена таблиця символів відповідно до ASCII кодів:

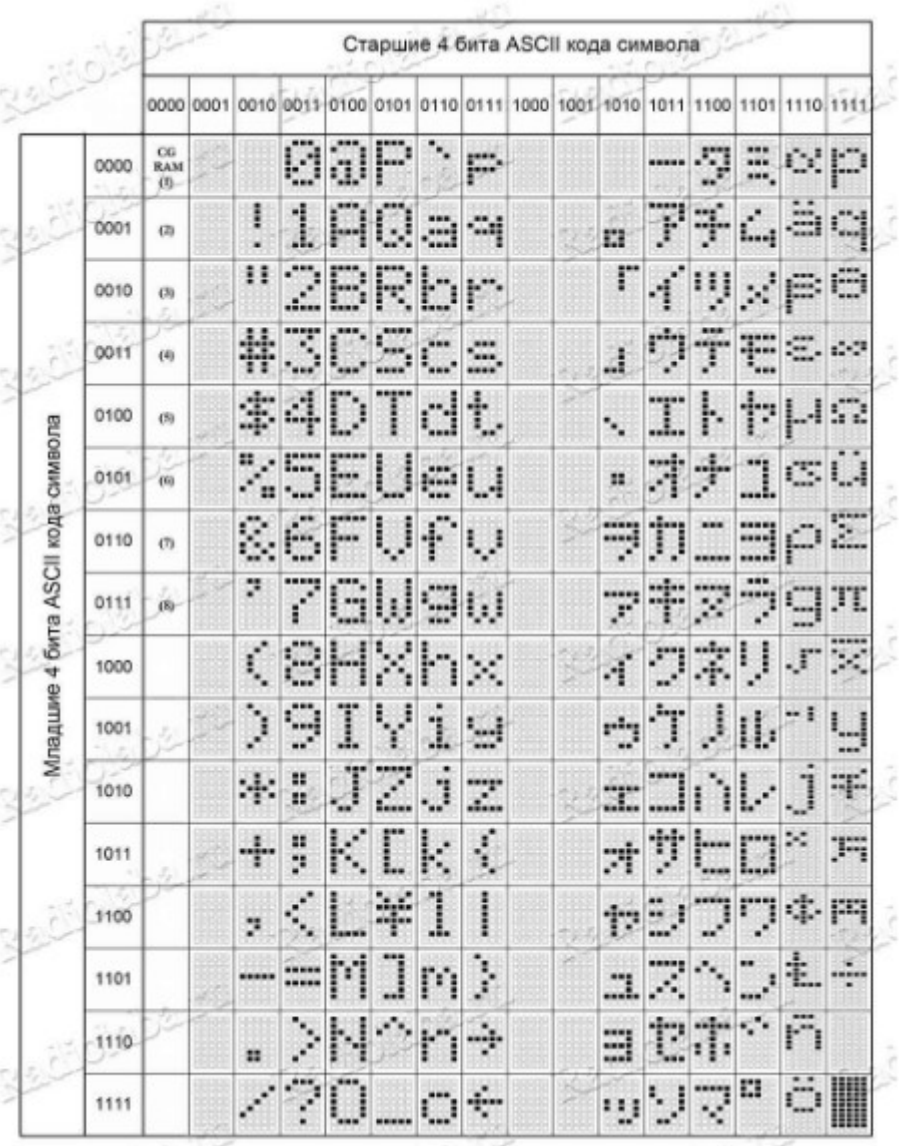

Рисунок 3.8 – Таблиця символів, доступних для виведення, і збережених в пам’яті CGROM.

Кожна інструкція виконується певний час, який вказано в наведеній вище таблиці (Рисунок 3.4), завершення внутрішньої операції можна дізнатися за допомогою читання прапора зайнятості BF (інструкція №9). Але зазвичай прапор не опитують, а просто витримують відповідну паузу після передачі інструкції. Взагалі, варто відмітити, що читання даних з дисплею застосовується рідко, і не розглядається в рамках даної роботи.

Для виведення символу необхідно записати адресу регістра DDRAM пам'яті в адресний покажчик (інструкція №8), тим самим вибравши положення

символу на екрані, потім записати в обраний регістр код ASCII символу (інструкція №10), виходячи з отриманого коду, контролер дисплея отримає дані з CGROM пам'яті для промальовування символу в заданому положенні на екрані. Після виведення символу, адресний покажчик автоматично інкрементується або декрементується, в залежності від того, яке значення було задано бітом напрямку I/D в відповідній інструкції. Таким чином, символи можна виводити послідовно, при цьому корегування адреси DDRAM пам'яті не потрібно. Наприклад, якщо перший рядок повністю заповниться символами, то відбудеться перехід на другий рядок, і навпаки.

Енергонезалежна пам'ять CGRAM (Character Generator RAM) призначена для створення унікальних символів за потреби розробника. Обсяг пам'яті невеликий, і дозволяє зберігати 8 довільних символів. Для створення одного символу розміром 5 × 8 пікселів, необхідно передати 8 байт даних в регістри пам'яті.

Перед створенням символу необхідно записати адресу регістра CGRAM пам'яті в адресний покажчик (інструкція №8), далі передати байти даних, які складуть вигляд символу (інструкція №10), адресний покажчик автоматично інкрементується або декрементується (в залежності від біта I/D), як і в разі DDRAM пам'яті. При створенні символу беруть участь тільки 5 молодших бітів байту даних. Символам присвоюються коди 0x00-0x07, відповідно до їх розташуванням в пам'яті CGRAM.

Варто відмітити, що даний режим роботи також не представляє інтересу в рамках даної роботи.

Ініціалізація дисплею LCD 1602.

Загалом, порядок ініціалізації наводиться в документації. В якості прикладу можна навести наступний алгоритм ініціалізації:

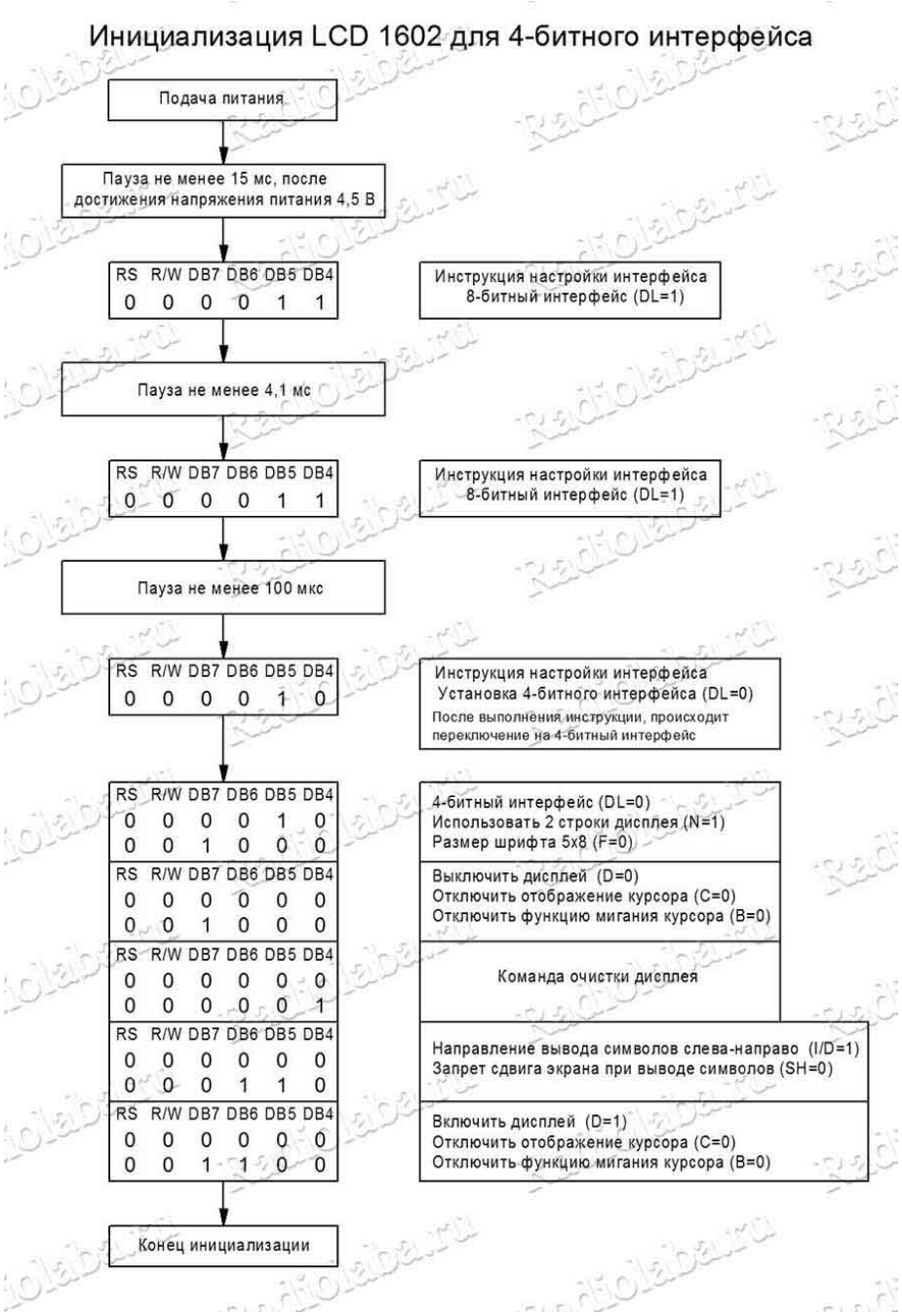

Рисунок 3.9 — Приклад порядку ініціалізації дисплею для роботи в режимі 4-бітного інтерфейсу.

Після ініціалізації згідно з алгоритмом, наведеним на Рисунку 3.8., дисплей налаштований на використання 2-х рядків, екран очищений від символів, курсор встановлений на початок першого рядка (адреса DDRAM пам'яті 0x00), відображення курсора відключено, обраний шрифт 5×8 пікселів,

напрямок виведення символів зліва-направо, зсув екрану при виведенні символів заборонений.

Якщо дозволити зсув екрану при виведенні символів (біт SH = 1), то з кожним новим символом екран буде зсуватись в обраному напрямку, тобто нові символи будуть з'являтися в заданому положенні на екрані, а інші зсуватимуться.

Додатково на екрані можна також включити відображення курсора (біт C = 1), який виглядає у вигляді лінії підкреслення, і може блимати в залежності від налаштувань. Положення курсору на екрані відповідає поточній адресі DDRAM пам'яті в адресному покажчику. Курсор можна переміщувати по екрану (інструкція №5), при цьому адресний покажчик буде інкрементуватись/ декрементуватись в залежності від напрямку зсуву.

Підключення дисплею типу 1602 через I2C інтерфейс

Спільно з цим дисплеєм може бути використаний модуль-перехідник на основі мікросхеми PCF8574AT, яка призначена для розширення кількості ліній вводу/виводу. Дана мікросхема підключається по I2C інтерфейсу і має порт з 8 ліній вводу/виводу. Принцип функціонування полягає в тому, що при записі байта даних в мікросхему, лінії порту приймають рівні, які відповідають значенням бітів отриманого байта. Операція читання повертає байт даних, біти якого відповідають стану ліній порту. Таким чином, мікросхема дозволяє розширити кількість ліній вводу/виводу, використовуючи дві керуючі лінії — SDA та SLC.

На наступному рисунку наведена типова схема підключення дисплею та модуля розширювача портів.

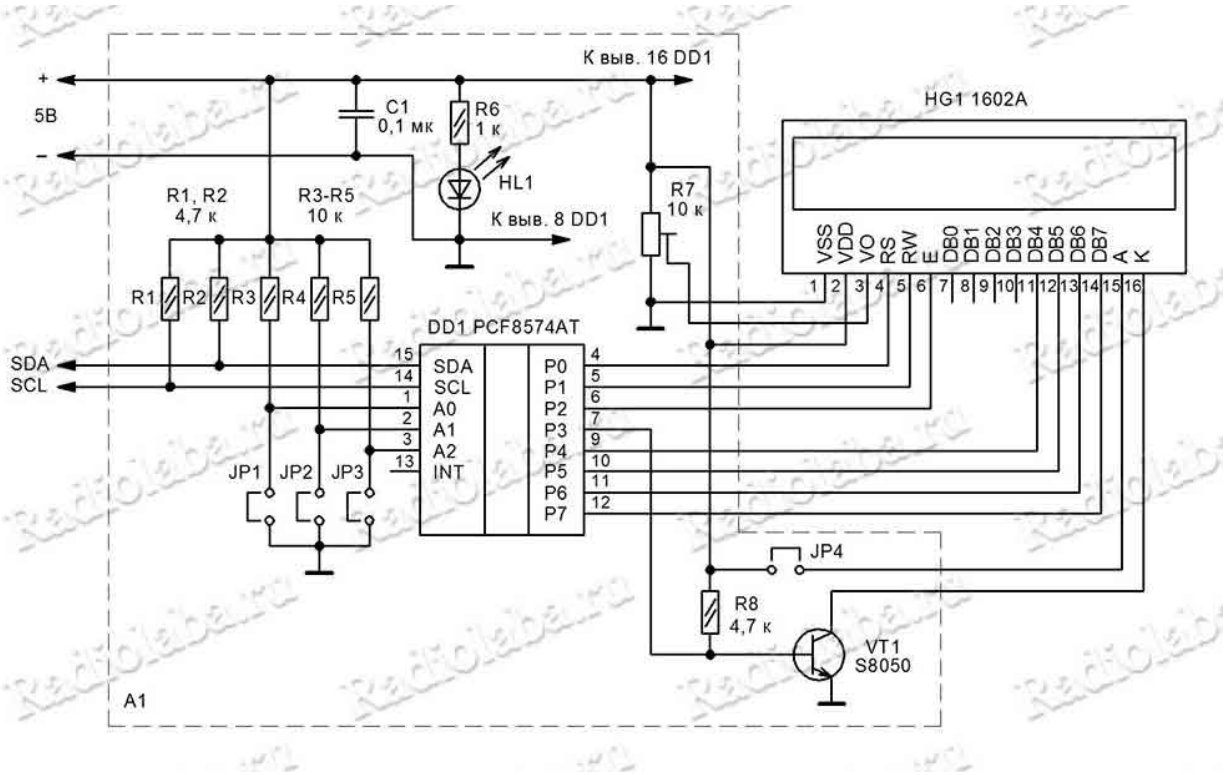

Рисунок 3.10 – Схема підключення дисплея та I2C-модуля.

Адресу мікросхеми PCF8574AT на шині I2C можна налаштовувати, при цьому старші 4 біти адреси є фіксованими, і рівні 0111, молодші 3 біти залежать від стану входів мікросхеми A2-A0. На модулі дані входи підтягнуті до високого рівня, відповідно адреса мікросхеми приймає значення 0111111.

Як видно зі схеми, наведеної вище, до мікросхеми підключена тільки частина ліній вводу/виводу дисплея DB7-DB4, це означає, що управління дисплеєм можливе тільки через 4-бітний інтерфейс. Для введення інструкції в дисплей потрібно 2 тактових імпульси на лінії E — тобто послідовність рівнів 1010 ("замикання" даних відбувається по спаду рівня), в результаті необхідно записати в мікросхему 4 байти для однієї інструкції.

На наступному рисунку наведено приклад запису інструкції в дисплей по інтерфейсу I2C.

Пример записи инструкции в LCD 1602 через I2C интерфейс (PCF8574AT)

Инструкция настройки интерфейса, количества активных строк, размера шрифта
4-битный интерфейс (DL=0), использовать 2 строки дисплея (N=1), размера шрифта 5x8 (F=0)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	-	-

Для ввода инструкции используется 4-битный интерфейс LCD 1602

P7	P6	P5	P4	P3	P2	P1	P0	Линии ввода/вывода PCF8574AT
DB7	DB6	DB5	DB4	LED	E	R/W	RS	Линии LCD 1602 (LED-подсветка дисплея)
0	0	1	0	0	1	0	0	1-й байт: запись старшего полубайта команды, линия тактирования E=1
0	0	1	0	0	0	0	0	2-й байт: запись старшего полубайта команды, линия тактирования E=0
1	0	0	0	0	1	0	0	3-й байт: запись младшего полубайта команды, линия тактирования E=1
1	0	0	0	0	0	0	0	4-й байт: запись младшего полубайта команды, линия тактирования E=0

Рисунок 3.11 – Приклад передачі інструкції керування дисплеєм LCD 1602 через інтерфейс I2C з використанням розширювача ліній вводу-виводу PCF8574AT.

Як видно, спочатку передається старший напівбайт інструкції з бітом E = 1, потім він же, але з бітом E = 0, при цьому в дисплей передається перша половина інструкції. Далі таким же чином передається друга половина (молодший напівбайт).

Для управління підсвічуванням дисплея, на платі модуля розміщений транзистор, підключений до лінії P3 мікросхеми. Таким чином, 3-й біт (при нумерації, починаючи з 0) в байті даних, який передаватиметься за допомогою I2C, керує підсвічуванням.

3.5. Опис алгоритму та принципу роботи з модулем RTC.

Модуль годинника реального часу в мікроконтролері STM32F401RE реалізований апаратно — всередині МК. Він володіє наступними можливостями:

1) автоматичне пробудження у всіх режимах енергозбереження;
2) незалежний BCD таймер-лічильник. Відлік часу і календар реалізовані апаратно, з можливістю настройки сигналізуючих переривань;
3) програмний «прапорець» пробудження з можливістю виклику переривання;
4) два 32-х розрядних регістра, в яких зберігаються секунди, хвилини, години (в 12 годинному або 24-годинному форматі), день тижня, день місяця, місяць і рік;
5) компенсація довжини місяця (а також високосного року) виконується автоматично;
6) два 32-х розрядних регістра програмованих сигналізуючих переривань;
7) наявність калібрування для компенсації відхилення кварцового резонатора;
8) після скидання область RTC захищена від випадкового запису;
9) до тих пір поки напруга живлення RTC залишається в робочому діапазоні, він буде працювати незалежно від поточного режиму роботи (Run mode, low-power mode або ж under reset);
10) є можливість записувати час виникнення події, так званий Time-stamp;
11) визначення втручання (tamper), при цьому скидаються всі backup-регістри;

Події, які можуть викликати переривання:

1) Alarm A;
2) Alarm B;
3) Wakeup interrupt;

4) Time-stamp;

5) Tamper detection;

Для безперервної роботи годинника, лічильникам RTC необхідне окреме джерело живлення, в якості якого найчастіше виступає батарейка, яка підключається до контакту VBAT мікроконтролера.

Для тактування модуля RTC може використовуватися одне з трьох джерел:

1) LSI — низькочастотний внутрішній RC-генератор на 37 KHz;
2) LSE — зовнішній, "годинниковий" кварцевий резонатор на 32768 Hz;
3) HSE (/2, 4, 8, 16) — зовнішній високочастотний кварцовий резонатор з відповідним переддільником;

Вибір джерела тактування налаштовується бітами поля RTCSEL [1: 0] в регістрі RCC_BDCR.

Після того, як джерело тактування було вибрано, необхідно потурбуватись про те, щоб рахункові регістри RTC тактувались з частотою 1Hz. Для зниження частоти тактування до 1Hz призначені два переддільники — синхронний та асинхронний. Розрахунок їх значень здійснюється за наступною формулою:

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

Рисунок 3.12 — Формула для розрахунку значень переддільників системи тактування модуля RTC.

При використанні значень PREDIV_A = 127 та PREDIV_S = 255 — з початкової частоти 32768 Hz буде отримано необхідну частоту тактування лічильників, рівну 1 Hz.

Для налашування часу виділено один 32-х бітний регістр, розбитий на бітові поля. Дані в ньому зберігаються в форматі BCD — одиниці і десятки в окремих бітових полях.

Також варто звернути увагу на біт PM, який відповідає за формат зберігання годин. Якщо в ньому встановлений нуль, вибрано буде 24-годинний формат відліку та збереження часу, якщо одиниця — 12-годинний.

Дата також зберігається в 32-розрядному регістрі, розбитому на бітові поля.

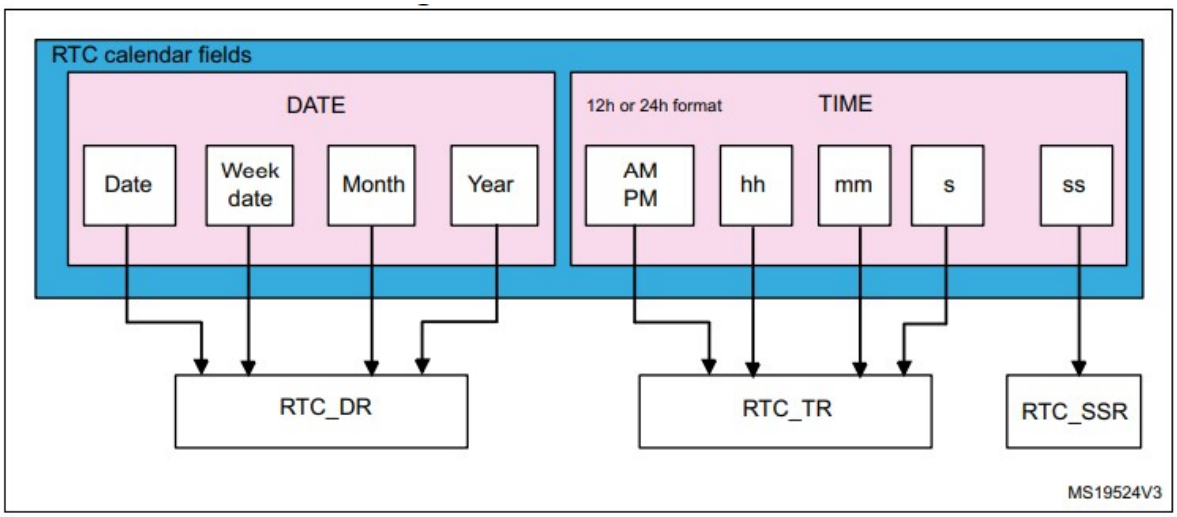

Рисунок 3.13 — Поля регістрів дати та часу модуля RTC.

Основні регістри модуля RTC, необхідні для початку роботи з ним, і котрі використовуються в програмі керування пристроєм, що розглядається, були описані вище, в Розділі 3, пункті 3.1.

3.6. Загальний опис алгоритму роботи програми.

Програма керування пристроєм починає своє виконання з функції main. Перш за все, відбувається конфігурація портів вводу-виводу — функція init_GPIO. При цьому відбувається дозвіл тактування порту B модулю GPIO мікроконтролера. Далі піни 13, 14 та 15 даного порту налаштовуються на вхід, а також вмикається внутрішня їх підтяжка до напруги живлення.

Далі виконується функція btn_irq_init, яка виконує налаштування переривань від зовнішніх джерел — вмикає тактування контролеру конфігурації системи, вибирає піни PB13, PB14, PB15 в якості джерел зовнішніх переривань, дозволяє переривання від вищеназваних джерел, налаштовує виникнення переривань по приходу заднього фронту сигналу на

відповідних пінах, які є джерелами зовнішніх переривань, а також, за допомогою відповідних функцій контролеру NVIC, дозволяє роботу обробника даних переривань (EXTI15_10_IRQn), очищує відповідні pending-біти, а також встановлює найвищий пріоритет для обробки даних переривань. Після цього відбувається глобальний дозвіл переривань.

Після цього відбувається конфігурація роботи системного таймера SysTick. Далі конфігурується робота таймера TIM2, за допомогою prescaler'а встановлюється режим роботи в 1000 тактів таймеру за секунду, а також переповнення таймера кожні 10 тактів (так званих «тіків» таймера). Також налаштовується дозвіл переривань по переповненню таймера, налаштовується режим рахування «вгору», відбувається дозвіл переривань від таймера TIM2, за допомогою відповідної функції контролера NVIC, а також глобально дозволяються переривання.

Після цього налаштовується робота модуля I2C1. Перш за все, дозволяється тактування шини APB1, яка зв'язана з даним модулем, далі виконується налаштування пінів PB8 та PB9 — виконується вибір альтернативної функції для даних пінів, а саме робота у ролі ліній SCL та SCK модуля I2C1. Також дані піни налаштовуються на роботу в режимі альтернативної функції, а також режимі відкритого стоку (open drain).

Що до конфігурації модуля I2C1, він налаштовується наступним чином: тактування модулю від частоти шини PB1, що рівна 16 МГц, генерація тактового сигналу на лінії SCL з частотою 100 кГц, режим роботи — стандартний. Також необхідним чином, згідно документації та вибраного режиму роботи — стандартний режим, розраховано було значення, яке заноситься в регістр TRISE, яке відповідає за налаштування «таймінгів» сигналу SCL, який генеруватиметься модулем I2C1. Після цього відбувається дозвіл роботи модулю I2C1.

Далі відбувається конфігурація роботи LCD дисплею. Дану процедуру виконує функція LCD1602_init. Процес конфігурації полягає в послідовній

відправці по інтерфейсу I2C необхідних команд керування контролером дисплея. Для цього, загалом, застосовуються функції для роботи з I2C з відповідної самописної бібліотеки lcd1602.h, написаної в рамках даної роботи для виконання даного завдання. При цьому перші кілька команд керування надсилаються за два «такти», наступні ж команди — за 4 «такти», тобто, у вигляді 4 байтів, надісланих по лінії даних SDA інтерфейсу I2C.

Послідовність команд керування дисплеєм — цілком стандартна, і наводиться виробником в документації. Загалом, після виконання процедури ініціалізації, дисплей налаштований на роботу в наступному режимі: використання 4-бітного інтерфейсу, використання 2 рядків для виведення символів, шрифт 5x8, курсор вимкнений, його «блимання» вимкнено, дисплей очищено, виведення символів — зліва направо, зсув дисплея при виведенні символів вимкнено, покажчик адреси DDRAM встановлено в значення 0x0.

Значення-команди, які надсилаються по I2C для конфігурації роботи дисплея, наведені в вихідному коді. «Декодувати» їх можна, співставивши з таблицями, наведеними на Рисунках 3.1 та 3.2.

Після цього відбувається перехід в нескінченний цикл while(1). В ньому програма весь час очікує на встановлення в значення «1» «прапорця» LCD_show_ready, який свідчить про те, що значення вхідних даних було оновлено, відповідно, з'являється потреба оновити дані, що виводяться на дисплей.

За умови, коли даний «прапорець» не рівний «1», контролер виконує «порожнюю інструкцію» — NOP.

Розглянемо алгоритм роботи програми за умови, коли «прапорець» LCD_show_ready встановлюється в «1».

В такому випадку, перш за все, здійснюється перевірка значення змінної-«прапорця» is_alarm, рівність якої значенню «1» свідчить про те, що відбулось спрацювання будильника. В такому випадку відбувається очищення вмісту

дисплею, виведення текстового рядка, активація світлодіоду HL1, котрий здійснює візуальну індикацію ситуації спрацювання будильника.

Якщо значення is_alarm рівне «0», здійснюється перевірка значення змінної clock_cfg_mode. За умови рівності даного значення «1», відбувається індикація режиму налаштування значень поточних дати та часу, а також виведення, при цьому, на екран та оновлення, за потреби, значень дати та часу, котрі, після закінчення їх налаштування користувачем, будуть збережені в відповідних регістрах дати та часу модуля RTC мікроконтролера.

Якщо значення clock_cfg_mode рівне «0», перевіряється значення змінної alarm_cfg_mode. За умови рівності значення даної змінної «1», відбудеться індикація режиму налаштування будильника, що здійснюється аналогічно описаній вище індикації режиму налаштування значень поточної дати та часу.

Якщо значення змінної alarm_cfg_mode рівне «0», відбувається перевірка значення змінної clock_show_mode. За умови його рівності «1», кожну секунду, що контролюється значенням змінної clk_1hz, котра встановлюється в «1» та переходить в «0» саме кожну секунду, відбувається запит та отримання поточних значень дати та часу у модуля RTC, і їх виведення на дисплей. При цьому, якщо при отриманні цих даних відбулась помилка, на екран буде виведено відповідне повідомлення про помилку отримання даних.

Якщо ж значення змінної clock_show_mode рівне «0», відбувається перевірка значення змінної alarm_show_mode. За умови його рівності «1», відбудеться запит в модуля RTC значень налаштувань дати та часу будильника, і, якщо ці дані будуть отримані, відбудеться їх відображення на екрані, з відповідною індикацією того, увімкнений будильник, чи ні. Якщо ж при отриманні цих даних відбудеться помилка, користувач буде сповіщеним про це відповідним повідомленням, що відобразиться на дисплеї.

Якщо ж значення змінної alarm_show_mode буде рівним «0», відбудеться «порожня інструкція» NOP.

Виведення відповідних даних на екран відбувається за допомогою функцій власноруч написаної бібліотеки для роботи з LCD-дисплеєм типу 1602 на основі контролера HD44780, а також розширювача портів вводу-виводу PCF8574.

Окремо варто зупинитись на роботі функції I2C_send, яка виконує відправку даних по I2C від пристрою-Master'а, яким виступає STM32F401RE, до пристрою-Slave'а, яким виступає PCF8574AT.

Формат передачі даних від пристрою-ведучого (Master) до пристрою-веденого (Slave) наведений в документації на мікроконтролер. Реалізується це, загалом, методом «поллінгу» та очікування встановлення потрібних значень бітів в регістрах статусу модуля IC21.

Перш за все, відбувається ініціалізація старт-біта. Далі відбувається очікування встановлення в «1» біту-«прапорця» SB в регістрі SR1модуля I2C1. Далі, за успішного виконання даної умови, відбувається надсилання адреси Slave'а по лінії даних, шляхом її запису в регістр DR. Після цього потрібно чекати, поки відбудеться встановлення в «1» біту ADDR в регістрі SR1, що свідчитиме про успішний прийом Slave`ом даної адреси та готовність до «спілкування» з пристроєм-ведучим. Далі, згідно документації, потрібно зчитати значення регістрів SR1 та SR2. Після цього відбувається надсилання необхідних байтів даних, шляхом їх почергового запису в регістр DR, і очікування, поки в данному регістрі не з'явиться вільне місце, про що сигналізує біт TXE в регістрі SR1. Коли наявні для відправки байти були відправлені, очікуємо встановлення в «1» значення біту BTF в регістрі SR1, що свідчить про закінчення передачі даних. Закінчується виконання даної функції формуванням стоп-біта.

Ключовою в роботі пристрою також є функція PCF8574AT_send, яка приймає на вхід 8-бітну команду керування дисплеєм, значення led_flag, яке визначає, буде увімкнена підсвітка дисплею, чи ні, значення rw_flag, що

визначає, буде відбуватись запис в дисплей, чи зчитування з нього, значення rs_flag, яке визначає, інструкція буде надіслана дисплею, чи дані, а також значення cycles — кількості циклів, за які буде передана команда, та значення delay_ms — затримки, яка буде здійснена після відправки команди.

Під кількістю «циклів» — значення cycles, мається на увазі необхідність «замикання» значення команди, яка була надіслана контролеру дисплея, заднім фронтом сигналу E. Тобто, потрібно надіслати команду спочатку з виставленим в високий рівень значенням сигналу «тактування» контролеру E, а тоді — надіслати, наприклад, її ж, але уже виставивши значення сигналу E в низький рівень. Це, в контексті розробленого алгоритму роботи — і є два «цикли».

Команди надсилання даних — ASCII-коду символів, наприклад, відбуваються за чотири «цикли» — спершу надсилаються, як уже згадувалось вище, чотири старші біти інструкції керування контролером дисплея (див. Рисунок 3.4), а також необхідні значення LED, E, R/!W та RS, згідно формату інструкцій PCF8547AT, з виставленим в високий рівень значенням біту E, далі надсилаються чотири молодші біти інструкції керування контролером дисплея, а після цього — це ж значення, але з виставленим в низький рівень значенням біту E, що призведе до «замикання» даного байту даних в пам’яті контролера дисплея.

Функція PCF8574AT_send, при цьому, перетворює інструкцію в форматі, наведеному в таблиці на Рисунку 3.4, яка передається в дану функцію у вигляді аргументу data, в формат, який підходить для передачі за допомогою PCF8574AT, переставляючи необхідним чином значення потрібних бітів, і формуючи «посилки» даних у вигляді, що підходить для надсилання по лінії даних I2C.

Наглядно відмінність в форматі команд керування контролером LCD дисплею, налаштованого на роботу в 4-бітному форматі команд керування, та форматі відповідних команд, які надсилаються з використанням PCF8574AT, відображена в наступній таблиці.

Таблиця 3.5 — Формати інструкцій керування контролером дисплея та «пакетів», що надсилаються до PCF8574AT.

Формат інструкції керування контролером LCD дисплея									
RS	R/!W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Формат «пакету», що надсилається по лінії даних SDA до PCF8547AT									
DB7/3	DB6/2	DB5/1	DB4/0	LED	E	R/!W	RS		

Як видно, «пакети», які надсилаються по лінії даних до PCF8547AT мікроконтролером, містять в старших чотирьох бітах чотири біти інструкції керування контролером дисплея, а в чотирьох молодших бітах — біти, які визначають увімкнення/вимкнення підсвітки дисплея, біт E — сигнал тактування, заднім фронтом якого відбувається «замикання» надісланої інструкції в пам’яті контролера дисплея, а також біти R/!W та RS, які визначають, відповідно, зчитані будуть дані з контролеру дисплея, чи записані в нього, а також те, командою керування є надіслані дані, чи даними для відображення на дисплей.

Як видно також, інструкція керування контролером дисплея містить 8 біт DB7-DB0, які, зокрема, і визначають її призначення та функцію. А «пакет», що надсилається по I2C до PCF8574AT містить при цьому, в свою чергу, чотири біти з цих 8 біт, які «кодують» інструкцію. Відповідно, в 4-бітному режимі роботи дисплею, з використанням PCF8574AT, потрібно надіслати 2 «пакети» даних, щоб коректно сформувати повністю інструкцію керування контролером дисплею — спочатку в старших 4 бітах «пакету» будуть міститись старші 4 біти інструкції керування контролером дисплея (DB7-DB5), а після цього, в наступному «пакеті», його старші 4 біти міститимуть уже молодші 4 біти інструкції (DB3-DB0). Крім цього, потрібно також врахувати необхідність «замикання» надісланих даних в пам’яті контролера дисплея, надсилаючи «пакети» з послідовним чергуванням рівнів сигналу E («1» → «0»). Відповідно,

в сумі на надсилання однієї 8-бітної інструкції керування контролером дисплея в 4-бітному режимі його роботи, потрібно надіслати 4 байти даних до PCF8574AT по лінії даних SDA — «пакет» з старшими 4 бітами (DB7-DB5) 8-бітної інструкції керування та сигналом E, встановленим в «1», цей же «пакет», але з сигналом E, встановленим в «0», «пакет» з молодшими 4 бітами (DB3-DB0) 8-бітної інструкції керування та сигналом E, встановленим в «1», цей же «пакет», але з сигналом E, встановленим в «0». Це — і є 4 «такти» (cycles), про які йшлося вище при поясненні принципу роботи функції PCF8574AT_send.

Після приведення інструкції до потрібного вигляду, сформовані дані передаються в функцію I2C_send, яка надсилає їх по I2C за вказану кількість «циклів», необхідну для коректного «замикання» надісланих даних в пам’яті контролера дисплея. Після цього викликається функція softdelay, яка реалізує необхідну часову затримку після надсилання інструкції.

4. Інструкція для користувача.

1. Завантажити програму керування пристроєм в Flash-пам'ять мікроконтролера STM32F401, що використовується в схемі пристрою, за допомогою наявних засобів.
2. Зібрати макет пристрою згідно схеми електричної принципової ДК61.403231.001Е3.
3. Забезпечити живлення схеми пристрою.
4. За допомогою відповідних, наявних у пристрої, засобів введення, здійснити початкове налаштування поточних дати та часу, а також, за бажанням — налаштування часу спрацювання сигналу будильника та його активацію.

Висновки

Отже, в рамках даного курсового проекту, було розроблено пристрій, що здійснює відображення значень реального часу з функціями календаря та будильника.

Основою пристрою є мікроконтролер STM32F401, встановлений на налагоджувальній платі STM32F401 Nucleo, введення даних відбувається за допомогою трьох тактових кнопок з самоповерненням, засоби індикації — рідкокристалічний дисплей типу 1602 на основі контролера HD44780, а також світлодіод.

В процесі роботи була створена схема електрична принципова, були проведені необхідні розрахунки, була створена програма керування пристроєм, згідно вимог, поставлених в технічному завданні.

Також була створена інструкція для користувача, а також підготовлений перелік необхідної конструкторської документації: схема електрична принципова, перелік елементів.

Список використаних джерел

1. Документація на мікроконтролер STM32F401RE/[Електронний ресурс] — Режим доступу: https://www.st.com/content/ccc/resource/technical/document/reference_manual/5d/b1/ef/b2/a1/66/40/80/DM00096844.pdf/files/DM00096844.pdf/jcr:content/translations/en.DM00096844.pdf — Дата звернення 26.12.2019.
2. Документація на налагоджувальну плату STM32F401 Nucleo/[Електронний ресурс] — Режим доступу: https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf — Дата звернення 26.12.2019.
3. Документація на LCD дисплей типу 1602 на основі контролера HD44780/[Електронний ресурс] — Режим доступу: https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf — Дата звернення 26.12.2019.
4. Документація на розширювач портів вводу-виводу на основі мікросхеми PCF8574AT/[Електронний ресурс] — Режим доступу: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf — Дата звернення 26.12.2019.
5. Інструмент для налаштування тактування мікроконтролеру STM32F401RE — Clock configuration tool for STM32F40x/41x microcontrollers (AN3988)/ [Електронний ресурс] — Режим доступу: https://my.st.com/content/my_st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stsw-stm32091.html — Дата звернення 26.12.2019.

6. Підключення LCD1602 по I2C інтерфейсу/[Електронний ресурс] — Режим доступу: https://radiolaba.ru/microcotrollers/podklyuchenie-lcd-1602-po-i2c-interfeysu.html/ — Дата звернення 26.12.2019.
7. I2C інтерфейс/[Електронний ресурс] — Режим доступу: https://radiolaba.ru/microcotrollers/i2c-interfeys.html/ — Дата звернення 26.12.2019.
8. Використання модуля RTC в мікроконтролерах STM32 серії F4/[Електронний ресурс] — Режим доступу: https://www.st.com/content/ccc/resource/technical/document/application_note/7a/9c/de/da/84/e7/47/8a/DM00025071.pdf/files/DM00025071.pdf/jcr:content/translations/en.DM00025071.pdf — Дата звернення 26.12.2019.
9. STM32 RTC, Calendar/[Електронний ресурс] — Режим доступу: https://hubstub.ru/stm32/179-stm32-rtc-calendar.html — Дата звернення 26.12.2019.
10. Підключення LCD1602 по I2C інтерфейсу/[Електронний ресурс] — Режим доступу: https://radiolaba.ru/microcotrollers/podklyuchenie-lcd-1602-po-i2c-interfeysu.html/ — Дата звернення 26.12.2019.

Додатки

Додаток А

Технічне завдання

1. Найменування та галузь використання.

Пристрій для відображення реального часу з функціями календаря та будильника. Пристрій належить до галузі побутової електроніки. Використовується для індикації значень поточного часу та дати, а також в якості будильника.

2. Підстава для розробки.

Підставою для розробки є завдання, що видане викладачем згідно навчального плану.

3. Мета та призначення розробки.

Метою є розробка компактного, зручного в керуванні пристрою для відображення поточних значень дати та часу, з функціями календаря та будильника, з можливістю відображення часу в 24-годинному форматі, а також можливістю керування та введення даних за допомогою тактових кнопок з самоповерненням, з невеликою кількістю компонентів, що забезпечить високу надійність, а також гарні масогабаритні параметри.

4. Джерело розробки.

Пристрій розробляється вперше.

5. Технічні вимоги.

5.1. Функціональні можливості пристрою.

Засоби введення пристрою мають бути представленими трьома тактовими кнопками з самоповерненням, за допомогою яких здійснюється введення даних, а також вибір режиму роботи пристрою. Засоби

відображення пристрою мають бути представлені рідкокристалічним дисплеєм типу 1602 на основі контролера HD44780, а також світлодіодом. Основою пристрою є мікроконтролер STM32F401, в якому наявний вбудований модуль RTC.

5.2. Технічні характеристики.

Пристрій повинен забезпечувати:

- відображення часу в 24-годинному форматі;
- зручне керування та вибір режиму роботи;
- можливість задати день місяця, а також час — години та хвилини, сигналу спрацювання будильника;

5.3. Вимоги до рівня уніфікації та стандартизації.

Для виготовлення пристрою передбачається максимальне застосування

стандартних, уніфікованих деталей та виробів.

5.4. Вимоги до безпеки експлуатації та обслуговування.

Керуватися загальними вимогами безпеки до апаратури низької напруги ГОСТ 12.2.007-75.

5.5. Вимоги до складу та параметрів технічних засобів.

Для виробництва пристрою використовують матеріали вітчизняного, а також імпортного виробництва.

5.6. Вимоги до умов експлуатації.

Кліматичне виконання УХЛ 4.1 по ГОСТ 15150-69.

5.7. Вимоги до транспортування та зберігання.

Група умов зберігання Л1 по ГОСТ 15150-69. Зберігати в зачинених, опалювальних та вентильованих приміщеннях, в яких забезпечуються наступні умови: температура повітря +5...+40°С, відносна вологість повітря 60% при 200°С (середньорічне значення), атмосферний тиск 84...106кПа. Транспортувати автомобільним, залізничним або авіаційним видами транспорту в спеціальній транспортній тарі.

6. Результати роботи.

6.1. Результати даної роботи можуть бути використані як вихідна документація по створенню прототипу пристрою та подальшого впровадження його в серійне виробництво.

6.2. Дана робота (звітна документація) після виконання надається на кафедру КЕОА для подальшого захисту й зберігання в якості навчальної документації.

7. Робота повинна містити:

- Пояснювальну записку з Додатками;
- Схему електричну принципову;
- Перелік елементів;

8. Порядок розгляду й приймання роботи.

Порядок розгляду й приймання роботи на загальних умовах, прийнятих на кафедрі КЕОА. Рецензування й прийняття роботи комісією на загальних умовах.

9. Економічні показники.

В рамках даного проекту не розглядаються.

10. Етапи розробки.

№	Найменування етапу курсового проекту	Строки виконання етапу проекту
1.	Створення схеми електричної принципової	14.10-2.11
2.	Опис структури пристрою і його окремих складових	4.11-12.11
3.	Обгрунтування вибору елементної бази	13.11-17.11
4.	Опис і розрахунок схеми електричної принципової	19.11-21.11
5.	Розробка та затвердження графічної частини проекту	22.11-03.12
6.	Алгоритм роботи програми	22.11-03.12
7.	Інструкція користувача	04.12-19.12

Додаток Б

Лістинг програми керування пристроєм

Файл «main.c»

```
#include <stdint.h>
#include "stm32f4xx.h"
#include "lcd1602.h"

#define frequency    16000000UL    // 16 MHz high-speed internal (RC)

/*     GLOBAL VARIABLES START                              */

uint8_t LCD_show_ready = 0, debounce_ms_enable = 0, pressure_flag;
uint16_t pressure_10_ms_ticks = 0;
uint32_t field = 0;
uint32_t alarm_fields_num = 3, clock_fields_num = 7;
uint32_t alarm_cfg_mode = 0, clock_cfg_mode = 0, alarm_show_mode = 0, clock_show_mode =
1;
uint32_t btn_pressure_start = 0, btn_pressure_done = 0;
uint8_t time_get_done = 0, alarm_get_done = 0;
uint8_t clk_1hz = 0;
uint8_t alarm_enable = 0, is_alarm = 0;

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;

volatile RTC_struct_full RTC_data_full_buff;
volatile RTC_struct_brief RTC_data_brief_buff;

/*     GLOBAL VARIABLES END                        */

/********************************************** SYSTICK
**********************************************/

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;
}

void init_systick(void)
{
	SysTick->CTRL = 0;                                          // disable SysTick
	SysTick->LOAD = frequency - 1;                                      // set the
initial reload value
	SysTick->VAL = 0;                                           // reset the curent
SysTick counter value

	SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk;
	SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk; // switch off

	// use proc.clock (1 = processor clock, 0 = external clock (HCLK/8 = 2 MHz)),
enable interrupts (1 = enable)
}

void SysTick_Handler(void)
{
	if(debounce_ms_enable)
	{
		debounce_ms_enable = 0;

		SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk; // SysTick off

		EXTI->IMR |= (EXTI_IMR_IM13 | EXTI_IMR_IM14 | EXTI_IMR_IM15);   //
enable disabled interrupts
	}
}

void systick_debounce_ms(uint32_t debounce_ms)
{
	// default value is 250 ms debounce
	uint32_t time_ms = (debounce_ms >= 1 && debounce_ms <= 1000) ? debounce_ms :
250;

	SysTick->VAL = 0;                                   // clear current CNT value
	SysTick->LOAD = ((frequency / 1000) * time_ms) - 1;      // (time * 10^-3 *
frequency) - 1 = (time * 10^-3 * 16 000 000) - 1 = (time * 16 000) - 1

	debounce_ms_enable = 1;
```

```
	SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;  // enable SysTick
}

/*****************************************************************************
*******************/

/********************************************** TIM2
*********************************************/
void tim2_init(void)
{
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;                    // TIM2 clock
	TIM2->PSC = 16000 - 1;                                        // 1000 ticks
per second - 1ms - prescaler
	TIM2->ARR = 10-1;                                      // T = 10 ms = 10
ticks -> {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} - auto-reload-register
	TIM2->DIER |= TIM_DIER_UIE;                            // TIM2 DMA/interrupt
enable register - allow events by timer -  Update interrupt enable
	TIM2->CR1 &= ~TIM_CR1_DIR;                             // upcounting, DIR = 0

	NVIC_EnableIRQ(TIM2_IRQn);   // enable interrupts by TIM2
	__enable_irq();      // global interrupts enable
}

// Кожні N мілісекунд в обробнику переривань таймера перевіряємо стан кнопки.

void btn_pressure_check(void)
{
	EXTI->IMR &= ~EXTI_IMR_IM13; // disable EXTI13 interrupts to make safe and clear
situation
	EXTI->IMR &= ~EXTI_IMR_IM14; // disable EXTI14 interrupts
	EXTI->IMR &= ~EXTI_IMR_IM15; // disable EXTI15 interrupts

	btn_pressure_start = 1;

	TIM2->CR1 |= TIM_CR1_CEN;                  // switch on TIM2 - 10 miliseconds to
the next TIM2 interrupt
}

/*****************************************************************************
*******************/

/********************************************** GPIO
*********************************************/

void init_GPIO(void)
{
	// OSPEEDR - 2 MHz by default after reset

	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;       // enable GPIOB

	GPIOB->MODER &= ~GPIO_MODER_MODE13;               // PB13 for input
	GPIOB->MODER &= ~GPIO_MODER_MODE14;               // PB14 for input
	GPIOB->MODER &= ~GPIO_MODER_MODE15;               // PB15 for input

	GPIOB->PUPDR |= GPIO_PUPDR_PUPD13_0;                    // pull-up for PB13 -
[01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD13_1;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPD14_0;                    // pull-up for PB14 -
[01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD14_1;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPD15_0;                    // pull-up for PB15 -
[01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD15_1;
}
```

```
/**************************************************************************************
*******************/

/************************************************ EXTI
*************************************************/

void Nucleo_LED_init(void)
{
	if (!(RCC->AHB1ENR & RCC_AHB1ENR_GPIOAEN))
	{
		RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;	// enable GPIOA
	}

	GPIOA->MODER |= GPIO_MODER_MODE5_0; // PA5 for output - Nucleo onboard LED
	GPIOA->MODER &= ~GPIO_MODER_MODE5_1;	// push-pull mode is by default
}

void btn_irq_init(void)
{
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;	// SYSCFG (System configuration
controller) clock through APB2 bus enable

	// interrupt source selection
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PB;	// PB13 -> EXTI13
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI14_PB;	// PB14 -> EXTI14
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI15_PB;	// PB15 -> EXTI15

	EXTI->IMR |= (EXTI_IMR_IM13 | EXTI_IMR_IM14 | EXTI_IMR_IM15);	// interrupt
request mask - IM11 and IM12 are not masked now
	EXTI->FTSR |= (EXTI_FTSR_TR13 | EXTI_FTSR_TR14 | EXTI_FTSR_TR15);	//
falling trigger

	NVIC_EnableIRQ(EXTI15_10_IRQn);
	NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
	NVIC_SetPriority(EXTI15_10_IRQn, 0);	// highest priority

	__enable_irq();	// enable interrupts, PRIMASK reset
}

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)
		{
			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;
		}
	}
}

/************************************************** RTC
**************************************************/

void RTC_auto_wakeup_enable(void)
{
	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

	// disable the wakeup timer
	RTC->CR &= ~RTC_CR_WUTE;

	// polling to make sure the access to wakeup auto-reload counter and to
WUCKSEL[2:0] bits is allowed
	while (!(RTC->ISR & RTC_ISR_WUTWF));

	// program the wakeup auto-reload value and the wakeup clock selection
	RTC->WUTR &= ~RTC_WUTR_WUT;  // the WUTF flag is set every (WUT[15:0] + 1) = (0
+ 1) = (1) ck_wut cycles
```

```
	RTC->CR &= ~RTC_CR_WUCKSEL_1;	// 10x: ck_spre (usually 1 Hz) clock is
selected
	RTC->CR |= RTC_CR_WUCKSEL_2;

	// enable the RTC Wakeup interrupt - enable the EXTI Line 22
	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))
	{
		RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
	}

	EXTI->IMR |= EXTI_IMR_IM22;  // interrupt request mask - IM22 is not masked now
	EXTI->RTSR |= EXTI_RTSR_TR22;	// rising edge trigger enabled for EXTI line
17

	NVIC_EnableIRQ(RTC_WKUP_IRQn);	// enable the RTC_WKUP IRQ channel in the
NVIC
	NVIC_ClearPendingIRQ(RTC_WKUP_IRQn);
	NVIC_SetPriority(RTC_WKUP_IRQn, 0); // highest priority

	// 1: Wakeup timer interrupt enabled
	RTC->CR |= RTC_CR_WUTIE;

	// enable the timer again
	RTC->CR |= RTC_CR_WUTE;

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;

	__enable_irq();	// global interrupts enable
}

void RTC_WKUP_IRQHandler(void)
{
	if(EXTI->PR & EXTI_PR_PR22)
	{
		if(RTC->ISR & RTC_ISR_WUTF)
		{
			RTC->ISR &= ~RTC_ISR_WUTF;   //  flag is cleared by software by
writing 0

			// GPIOA->ODR ^= GPIO_ODR_OD5;

			if (clock_show_mode)
			{
				clk_1hz = 1;
				LCD_show_ready = 1;
			}

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


void RTC_data_init(void)
{
	// uint32_t time_value, date_value;

	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

	// initialization mode on (INITF == 1) - calendar counter is stopped, can update
now
	RTC->ISR |= RTC_ISR_INIT;
```

```
	// INITF polling
	while (!(RTC->ISR & RTC_ISR_INITF));

	// prescalers - two separate write access - synch predivider
	RTC->PRER |= 0xFF; // 255

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

	RTC->TR = 0x00000000;
	RTC->DR = 0x00000000;

	// 24h format == 0
	RTC->CR &= ~RTC_CR_FMT;

	// exit from the init mode
	RTC->ISR &= ~RTC_ISR_INIT;

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;
}

void RTC_data_update(RTC_struct_full volatile *f_data)
{
	uint32_t time_value, date_value;
	time_value = time_value ^ time_value;
	date_value = date_value ^ date_value;

	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

	// initialization mode on (INITF == 1) - calendar counter is stopped, can update
now
	RTC->ISR |= RTC_ISR_INIT;

	// INITF polling
	while (!(RTC->ISR & RTC_ISR_INITF));

	// prescalers - two separate write access - synch predivider
	RTC->PRER |= 0xFF; // 255

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

	RTC->TR = 0x00000000;

	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));

	RTC->TR = time_value;

	RTC->DR = 0x00000000;

	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));

	RTC->DR = date_value;
```

```
	// 24h format == 0
	RTC->CR &= ~RTC_CR_FMT;

	// exit from the init mode
	RTC->ISR &= ~RTC_ISR_INIT;

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;
}

void RTC_alarm_init(void)
{
	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

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

	// wait for Alarm A write flag, to make sure the access to alarm reg is allowed
	while (!(RTC->ISR & RTC_ISR_ALRAWF));

	// date, hours, minutes, seconds mask - Alarm A set if they match
	RTC->ALRMAR &= ~RTC_ALRMAR_MSK4;    // 0: Alarm A set if the date/day match
	RTC->ALRMAR &= ~RTC_ALRMAR_MSK3;    // 0: Alarm A set if the hours match
	RTC->ALRMAR &= ~RTC_ALRMAR_MSK2;    // 0: Alarm A set if the minutes match
	RTC->ALRMAR |= RTC_ALRMAR_MSK1;            // 1: Seconds don’t care in Alarm A
comparison

	RTC->ALRMAR &= ~RTC_ALRMAR_WDSEL;   // DU[3:0] field represents the date units
	RTC->ALRMAR &= ~RTC_ALRMAR_PM;             // 0: AM or 24-hour format

	// enable the RTC Alarm interrupt - enable the EXTI Line 17
	// SYSCFG (System configuration controller) clock through APB2 bus enable
	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))
	{
		RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
	}

	EXTI->IMR |= EXTI_IMR_IM17;  // interrupt request mask - IM17 is not masked now
	EXTI->RTSR |= EXTI_RTSR_TR17;       // rising edge trigger enabled for EXTI line
17

	NVIC_EnableIRQ(RTC_Alarm_IRQn);     // enable the RTC_Alarm IRQ channel in the
NVIC
	NVIC_ClearPendingIRQ(RTC_Alarm_IRQn);
	NVIC_SetPriority(RTC_Alarm_IRQn, 0);       // highest priority

	// 1: Alarm A interrupt enabled
	RTC->CR |= RTC_CR_ALRAIE;

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;

	__enable_irq();       // global interrupts enable
}

void RTC_alarm_update(RTC_struct_full volatile *f_data)
{
	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

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

```
	// wait for Alarm A write flag, to make sure the access to alarm reg is allowed
	while (!(RTC->ISR & RTC_ISR_ALRAWF));

	RTC->ALRMAR |= (f_data->date_tens << RTC_ALRMAR_DT_Pos); // Bits 29:28 DT[1:0]:
Date tens in BCD format
	RTC->ALRMAR |= (f_data->date_units << RTC_ALRMAR_DU_Pos);// Bits 27:24 DU[3:0]:
Date units or day in BCD format.
	RTC->ALRMAR |= (f_data->hour_tens << RTC_ALRMAR_HT_Pos); // Bits 21:20 HT[1:0]:
Hour tens in BCD forma
	RTC->ALRMAR |= (f_data->hour_units << RTC_ALRMAR_HU_Pos);// Bits 19:16 HU[3:0]:
Hour units in BCD format.
	RTC->ALRMAR |= (f_data->minute_tens << RTC_ALRMAR_MNT_Pos);     // Bits 14:12
MNT[2:0]: Minute tens in BCD format.
	RTC->ALRMAR |= (f_data->minute_units << RTC_ALRMAR_MNU_Pos);    // Bits 11:8
MNU[3:0]: Minute units in BCD format.

	// enable Alarm A
	RTC->CR |= RTC_CR_ALRAE;

	// enable the RTC Alarm interrupt - enable the EXTI Line 17
	// SYSCFG (System configuration controller) clock through APB2 bus enable
	if (!(RCC->APB2ENR & RCC_APB2ENR_SYSCFGEN))
	{
		RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
	}

	EXTI->IMR |= EXTI_IMR_IM17;  // interrupt request mask - IM17 is not masked now
	EXTI->RTSR |= EXTI_RTSR_TR17;       // rising edge trigger enabled for EXTI line
17

	NVIC_EnableIRQ(RTC_Alarm_IRQn);     // enable the RTC_Alarm IRQ channel in the
NVIC
	NVIC_ClearPendingIRQ(RTC_Alarm_IRQn);
	NVIC_SetPriority(RTC_Alarm_IRQn, 0);       // highest priority

	// 1: Alarm A interrupt enabled
	RTC->CR |= RTC_CR_ALRAIE;

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;

	__enable_irq();      // global interrupts enable
}

void RTC_alarm_disable(void)
{
	alarm_enable = 0;     // clear enable flag

	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

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

	// wait for Alarm A write flag, to make sure the access to alarm reg is allowed
	while (!(RTC->ISR & RTC_ISR_ALRAWF));

	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.

	RTC->WPR = 0xFF;
}

void RTC_Alarm_IRQHandler(void)
{
	is_alarm = 1;  // display

	// unlock write protection
	RTC->WPR = 0xCA;
	RTC->WPR = 0x53;

	// disable Alarm A
	RTC->CR &= ~RTC_CR_ALRAE;
	alarm_enable = 0;

	// wait for Alarm A write flag, to make sure the access to alarm reg is allowed
	while (!(RTC->ISR & RTC_ISR_ALRAWF));

	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.

	// lock write protection - writing a wrong key reactivates the write protection
	RTC->WPR = 0xFF;

	RTC->ISR &= ~RTC_ISR_ALRAF;  //  flag is cleared by software by writing 0
	EXTI->PR |= EXTI_PR_PR17;    // clear pending flag
}

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);
}

/*************************************************************************************
************************/

void EXTI15_10_IRQHandler(void)
{
	EXTI->IMR &= ~EXTI_IMR_IM13; // disable EXTI13 interrupts
	EXTI->IMR &= ~EXTI_IMR_IM14; // disable EXTI14 interrupts
	EXTI->IMR &= ~EXTI_IMR_IM15; // disable EXTI15 interrupts

	systick_debounce_ms(500);    // start debouncing - you can't input some button
values more often, than 1 time in 500 ms

	if(EXTI->PR & EXTI_PR_PR13)  // PB13 - VALUE++            // !!! if((EXTI ->PR &
EXTI_PR_PR13) && (!(debounce_ms_enable))) -- WRONG
	{
		btn_fill_date_fields(&RTC_data_brief_buff, field, 1, alarm_cfg_mode,
clock_cfg_mode);

		EXTI->PR |= EXTI_PR_PR13;    // clear pending flag by writing 1 (clear
event flag after work)
	}
	else if(EXTI->PR & EXTI_PR_PR14)    // PB14 - VALUE--
	{
		btn_fill_date_fields(&RTC_data_brief_buff, field, (-1), alarm_cfg_mode,
clock_cfg_mode);

		EXTI->PR |= EXTI_PR_PR14;    // clear pending flag by writing 1
	}
	else if(EXTI->PR & EXTI_PR_PR15)    // PB15 - ALARM CONFIGURATON
	{

		// set pressure flag, buffer:
		pressure_flag = (!(GPIOB->IDR & GPIO_IDR_ID15));  // PB15 == 0 -->
pressure_flag = 1


		btn_pressure_check();        // save PB15 IDR value --> start TIM2 10ms
counting

		EXTI->PR |= EXTI_PR_PR15;    // clear pending flag by writing 1
	}

	LCD_show_ready = 1;
}

void TIM2_IRQHandler(void)
{
	TIM2->SR &= ~TIM_SR_UIF;     // clear pending bit in the interrupt handler

	// periodic checking
	if (btn_pressure_start)
	{
		if ((!(GPIOB->IDR & GPIO_IDR_ID15)) == pressure_flag)
		{
			pressure_10_ms_ticks = pressure_10_ms_ticks + 1;
```

```
                }
                else
                {
                        btn_pressure_start = 0;

                        TIM2->CR1 &= ~TIM_CR1_CEN;   // switch off TIM2

                        if (pressure_10_ms_ticks > 700)
                        {
                                RTC_alarm_disable();
                        }
                        else if (pressure_10_ms_ticks > 500 && pressure_10_ms_ticks <=
700)
                        {
                                // 700 x 10ms = 7000ms = 7s --> calendar-clock
configuration mode ON

                                clock_cfg_mode = 1;
                                alarm_cfg_mode = 0;

                                field = 0;
                        }
                        else if (pressure_10_ms_ticks >= 200 && pressure_10_ms_ticks <=
500)
                        {
                                // 200 x 10ms = 2000ms = 2s --> alarm configuration mode
ON (2s <= time <= 7s)

                                alarm_cfg_mode = 1;
                                clock_cfg_mode = 0;

                                field = 0;
                        }
                        else
                        {
                                // <2s press
                                if (clock_cfg_mode || alarm_cfg_mode)
                                {
                                        field = field + 1;
                                }
                                else if (is_alarm)
                                {
                                        is_alarm = 0; // short buttons pressing
                                }
                                else
                                {
                                        clock_show_mode = (clock_show_mode) ? 0 : 1;
                                        alarm_show_mode = (!(clock_show_mode));
                                }
                        }

                        pressure_10_ms_ticks = 0;    // reset cnt

                        if (alarm_cfg_mode)
                        {
                                if (field >= alarm_fields_num)
                                {
                                        fill_RTC_struct_full(&RTC_data_brief_buff,
&RTC_data_full_buff, 0);     // save final values

                                        RTC_alarm_update(&RTC_data_full_buff);     //
update

                                        // configuration done
                                        alarm_cfg_mode = 0;
```

```
                                        field = 0;

                                        alarm_enable = 1;
                                        alarm_show_mode = 1;  // ready to display final
alarm config data
                                        clock_show_mode = 0;
                                }
                        }
                        else if (clock_cfg_mode)
                        {
                                if (field >= clock_fields_num)
                                {
                                        // save final values
                                        fill_RTC_struct_full(&RTC_data_brief_buff,
&RTC_data_full_buff, 1);     // save final values

                                        RTC_data_update(&RTC_data_full_buff);      //
update


                                        // configuration done
                                        clock_cfg_mode = 0;
                                        field = 0;

                                        clock_show_mode = 1;  // ready to show final clock
data
                                        alarm_show_mode = 0;
                                }
                        }

                        EXTI->IMR |= (EXTI_IMR_IM13 | EXTI_IMR_IM14 | EXTI_IMR_IM15);
        // enable disabled interrupts
                }
        }

        LCD_show_ready = 1;
}

void RTC_get_time(RTC_struct_brief volatile *br_data)
{
        // we need to clear less bits: (RTC->DR & RTC_DR_DT)
        // and to shift right the part, which we want to --> to normal decimal

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

        uint32_t TR_buf = 0, DR_buf = 0;

        TR_buf = (RTC->TR);

        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));

        DR_buf = (RTC->DR);

        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);
```

```
	time_get_done = 1;
}

void RTC_get_alarm(RTC_struct_brief volatile *br_data)
{
	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));

	alarm_get_done = 1;
}

void LCD_clock_display(RTC_struct_brief volatile *br_data, uint32_t alarm_show,
uint32_t alarm_cfg, uint32_t clk_show, uint32_t clk_cfg)
{
	// 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F
	// 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F

	uint8_t *week_days[] = {"MON", "TUE", "WED", "THU", "FRI", "SAT", "SUN"};

	LCD1602_display_clear();

	if (alarm_show || alarm_cfg)
	{
		LCD1602_send_string("Alarm|Date: ", 0x00);
		LCD1602_send_string("|Time: ", 0x45);
		LCD1602_send_string(":", 0x4D);

		LCD1602_send_integer(br_data->date, 0x0C, 2);
		LCD1602_send_integer(br_data->hours, 0x4B, 2);
		LCD1602_send_integer(br_data->minutes, 0x4E, 2);

		if (alarm_cfg)
		{
			LCD1602_send_string("=CFG", 0x40);
		}
	}
	else if (clk_show || clk_cfg)
	{
		LCD1602_send_string(week_days[br_data->week_day-1], 0x02);
		LCD1602_send_integer(br_data->hours, 0x06, 2);
		LCD1602_send_string(":", 0x08);
		LCD1602_send_integer(br_data->minutes, 0x09, 2);
		LCD1602_send_string(":", 0x0B);
		LCD1602_send_integer(br_data->seconds, 0x0C, 2);

		LCD1602_send_integer(br_data->date, 0x46, 2);
		LCD1602_send_string(".", 0x48);
		LCD1602_send_integer(br_data->months, 0x49, 2);
		LCD1602_send_string(".", 0x4B);
		LCD1602_send_integer(br_data->years, 0x4C, 2);

		if (clk_cfg)
		{
			LCD1602_send_string("=CFG", 0x41);
		}
	}
}

void RTC_init(void)
{
```

```
	// check the INITS status flag in RTC_ISR register to verify if the calendar is
already initialized
	if (RTC->ISR & RTC_ISR_INITS)
	{
		return;
	}

	// PWR clock on
	RCC->APB1ENR |= RCC_APB1ENR_PWREN;

	// enable WRITE - allow access to backup registers (BDCR)
	PWR->CR |= PWR_CR_DBP;

	// pass only at the first time
	if (!(RCC->BDCR & RCC_BDCR_RTCEN))
	{
		// software reset - 1: Resets the entire Backup domain
		RCC->BDCR |= RCC_BDCR_BDRST;
		RCC->BDCR &= ~RCC_BDCR_BDRST;
	}

	// enable LSE - Low-speed external oscillator
	RCC->BDCR |= RCC_BDCR_LSEON;

	// wait for being ready by polling
	while (!(RCC->BDCR & RCC_BDCR_LSERDY));

	// LSE as clk source - [01] - LSE oscillator clock used as the RTC clock
	RCC->BDCR |= RCC_BDCR_RTCSEL_0;
	RCC->BDCR &= ~RCC_BDCR_RTCSEL_1;

	// RTC clock on
	RCC->BDCR |= RCC_BDCR_RTCEN;

	RTC_auto_wakeup_enable();
	RTC_alarm_init();

	// RTC_data_update(f_data);
	RTC_data_init();
}

void LED_indication(uint8_t enable)
{
	if (enable)
	{
		GPIOA->ODR |= GPIO_ODR_OD5;
	}
	else
	{
		GPIOA->ODR &= ~GPIO_ODR_OD5;
	}
}

/*****************************************************************************************
******************/

/********************************************** main loop
**********************************************/

int main(void)
{
	init_GPIO();
	btn_irq_init();
	init_systick();
	tim2_init();
```

```
	Nucleo_LED_init();

	I2C1_init();
	LCD1602_init();

	fill_struct_default(&RTC_data_brief_buff, &RTC_data_full_buff); // initial
structs filling

	RTC_init();
	RTC_data_update(&RTC_data_full_buff);

	while(1)
	{
		if(LCD_show_ready)
		{
			if (is_alarm)
			{
				// alarm situation handling
				// ends by any button short pressing

				LCD1602_display_clear();
				LCD1602_send_string("Wake up, Neo...", 0x00);
				LED_indication(is_alarm);
			}
			else if (clock_cfg_mode)
			{
				LCD_clock_display(&RTC_data_brief_buff, alarm_show_mode,
alarm_cfg_mode, clock_show_mode, clock_cfg_mode);
			}
			else if (alarm_cfg_mode)
			{
				LCD_clock_display(&RTC_data_brief_buff, alarm_show_mode,
alarm_cfg_mode, clock_show_mode, clock_cfg_mode);
			}
			else if (clock_show_mode)
			{
				if (clk_1hz)
				{
					// refresh every 1 second
					RTC_get_time(&RTC_data_brief_buff);

					if (time_get_done)
					{
						LCD_clock_display(&RTC_data_brief_buff,
alarm_show_mode, alarm_cfg_mode, clock_show_mode, clock_cfg_mode);

						time_get_done = 0;
					}
					else
					{
						LCD1602_display_clear();
						LCD1602_send_string("Time getting err",
0x00);
					}

					clk_1hz = 0;
				}
			}
			else if (alarm_show_mode)
			{
				RTC_get_alarm(&RTC_data_brief_buff);	// alarm time
configuration-initiaization

				if (alarm_get_done)
```

```
                                {
                                        LCD_clock_display(&RTC_data_brief_buff,
        alarm_show_mode, alarm_cfg_mode, clock_show_mode, clock_cfg_mode);

                                        if (alarm_enable)
                                        {
                                                LCD1602_send_string("=ON", 0x41);
                                        }
                                        else
                                        {
                                                LCD1602_send_string("=OFF", 0x41);
                                        }

                                        alarm_get_done = 0;
                                }
                                else
                                {
                                        LCD1602_display_clear();
                                        LCD1602_send_string("Alrm getting err", 0x00);
                                }
                        }

                        LCD_show_ready = 0;
                }
                else
                {
                        __NOP;
                }
        }
}
```

Файл «lcd1602.h»

```
#ifndef lcd1602_h
#define lcd1602_h

#include <stdint.h>

void softdelay(volatile unsigned long N);

void I2C1_init(void);
void I2C_send(uint8_t *pack, uint8_t pack_num);
void PCF8574AT_send(uint8_t data, uint8_t led_flag, uint8_t rw_flag, uint8_t rs_flag,
uint8_t cycles, uint32_t delay_ms);

void LCD1602_backlight_on(void);
void LCD1602_backlight_off(void);
void LCD1602_send_char(uint8_t character);
void LCD1602_init(void);
void LCD1602_set_cursor(uint8_t position);
void LCD1602_send_char_position(uint8_t character, uint8_t position);
void LCD1602_position_rst(void);
void LCD1602_display_clear(void);
void LCD1602_send_string(uint8_t *string, uint8_t position);
void LCD1602_send_integer(uint16_t number, uint8_t position, uint8_t sign_num);

void LCD1602_send_arith_results(uint8_t X, uint8_t Y, uint8_t mode, uint32_t result);
void arith_display(uint8_t X, uint8_t Y, uint8_t mode);
#endif
```

Файл «lcd1602.c»

```
#include "lcd1602.h"
#include <stdint.h>
#include "stm32f4xx.h"

#define frequency    16000000UL           // 16 MHz high-speed internal (RC)

/*     I2C CODE DEFINES && VARIABLES && MACRO START      */

#define I2C_MODE_TRANSMIT 0
#define I2C_ADDR(address, mode)     (((address) << 1) | (mode))  // [[addr[7:0]],[R/!W
bit]]

#define PCF8574AT_ADDRESS 0x3F      // tested with Arduino

#define LCD_READ 1    // R / !W
#define LCD_WRITE 0   // R / !W
#define LCD_INSTR 0   // RS
#define LCD_DATA 1    // RS
#define ENA 1 // latching
#define N_ENA 0       // latching

/*     DISPLAY INSTRUCTIONS                        */

#define INTERFACE_8_BIT 0x30              // 0011 XXXX -> 001 [DATA 8/4] XXXX
#define INTERFACE_4_BIT_2_ROWS_FONT_5X8 0x28      // 0010 10XX -> 001 [DATA 8/4]
[Number of lines 1/2] [Font size] XX
#define INTERFACE_4_BIT      0x20
#define DISPLAY_ON_CURSOR_ON_BLINK_DISABLE 0xE    // 0000 1110 -> 0000 1 [Display
on/off] [Cursor] [Blink]
#define DISPLAY_ON_CURSOR_ON_BLINK_ENABLE 0xF     // 0000 1111 -> 0000 1 [Display
on/off] [Cursor] [Blink]
#define DISPLAY_CLEAR 0x1                 // 0000 0001
#define DIRECTION_L_TO_R_SHIFT_DISABLE 0x6 // 0000 0110 -> 0000 01 [DDRAM address
Increment/Decrement] [Shift]
#define DDRAM_ADDRESS_0X0 0x80
#define DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE 0xC
#define DISPLAY_OFF_CURSOR_OFF_BLINK_DISABLE 0x8

/*     I2C CODE DEFINES && VARIABLES && MACRO END */

void softdelay(volatile unsigned long N)
{
       volatile unsigned long inner;

       while (N--)
       {
              for (inner = 100; inner > 0; inner--);
       }
}

// I2C initialization

void I2C1_init(void)
{
       RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;

       RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;       // enable GPIOB

       GPIOB->AFR[1] |= (4 << (4 * 0));    // PB8 - I2C1_SCL - for AF4
       GPIOB->AFR[1] |= (4 << (4 * 1));    // PB9 - I2C1_SDA - for AF4
```

```
	GPIOB->MODER &= ~(GPIO_MODER_MODER9 |GPIO_MODER_MODER8); // AF open drain mode,
external pull-ups
	GPIOB->MODER |= (GPIO_MODER_MODER9_1 |  GPIO_MODER_MODER8_1);
	GPIOB->OTYPER |=(GPIO_OTYPER_OT_9 |GPIO_OTYPER_OT_8);

	I2C1->CR2 &= ~I2C_CR2_FREQ;
	I2C1->CR2 |= 16;             // 16 MHz - PB1

	I2C1->CCR &= ~I2C_CCR_CCR;
	I2C1->CCR |= 80;             // 100 kHz SCL frequency

	I2C1->CCR &= ~I2C_CCR_FS;

	I2C1->TRISE = 17;            // (1000ns / Tpclk1) + 1

	I2C1->CR1 |= I2C_CR1_PE;     // peripheral enable
}

void I2C_send(uint8_t *pack, uint8_t pack_num)
{
	// Sends N = pack_num bytes of the data to the Slave.
	// Check "Figure 164. Transfer sequence diagram for master transmitter" to see
the events flow.

	I2C1->CR1 |= I2C_CR1_START;                // start bit
	while (!(I2C1->SR1 & I2C_SR1_SB));         // wait for SB (event 5)
	(void) I2C1->SR1;                          // clear SB flag - EV5: SB = 1,
cleared by reading SR1 register followed by writing DR
                                               // register with Address
	I2C1->DR = 0x7E;
	//I2C1->DR = 0x4E;
	while (!(I2C1->SR1 & I2C_SR1_ADDR));              // wait for ADDR
	(void) I2C1->SR1;                          // clear ADDR flag - EV6: ADDR = 1,
cleared by reading SR1 register followed by
	(void) I2C1->SR2;                          // reading SR2

	I2C1->DR = *pack;
	while (!(I2C1->SR1 & I2C_SR1_TXE));        // wait for data register empty

	pack++;

	for(uint8_t num = 0; num < pack_num-1; num++)
	{
		I2C1->DR = *pack++;                // send and go to the next cell of
the input array
		while (!(I2C1->SR1 & I2C_SR1_TXE));
	}

	while (!(I2C1->SR1 & I2C_SR1_BTF));        // byte transfer finished

	I2C1->CR1 |= I2C_CR1_STOP;                 // stop bit
}

// implementation of latching inputting way
void PCF8574AT_send(uint8_t data, uint8_t led_flag, uint8_t rw_flag, uint8_t rs_flag,
uint8_t cycles, uint32_t delay_ms)
{
	// latching timings (delays) were not taken in account

	// RS | R/!W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0     - instruction
format from datasheet
	// DB7 | DB6 | DB5 | DB4 | LED | E | R/!W | RS                   - I2C
(PCF8547AT instruction format)
```

```
	// cycles - 2 (instr -> latched_instr) or 4 (MSB_instr -> latched_MSB_instr ->
LSB_instr -> latched_LSB_instr)
	// RS: 0 = instr, 1 = data
	// rw - R / !W

	uint8_t led, ena, rw, rs, data_msbits, data_lsbits;	// data's 4 most
significant bits, and 4 least significant bits

	led = led_flag << 3;				// LED: 0 - backlight OFF, 1 -
backlight ON
	ena = ENA << 2;
	rw = rw_flag << 1;				// R/!W: 0 - write, 1 - read
	rs = rs_flag;					// RS: 0 - instruction, 1 -
data

	data_msbits = data & 0xF0;			// 1111 1111 & F0 = 1111 0000
	data_lsbits = (data << 4) & 0xF0;		// 1111 0001 << 4 = 0001 0000

	uint8_t package[4];

	for(uint8_t cells = 0; cells < 4; cells++)
	{
		package[cells] = 0;
	}

	package[0] = data_msbits | led | ena | rw | rs;		// ENA = 1
	package[1] = package[0] ^ 4;				// ENA = 0 -->
LATCHED!
	package[2] = data_lsbits | led | ena | rw | rs;		// ENA = 1
	package[3] = package[2] ^ 4;			// ENA = 0 --> LATCHED!

	I2C_send(package, cycles);			// send ready instruction

	softdelay(delay_ms);				// wait some time
	//st_delay_ms(delay_ms);				// required delay
after instruction was sent
	//while(~(st_delay_ms_rdy));			// yep, just wait here
}

void LCD1602_backlight_on(void)
{
	PCF8574AT_send(0x0, 1, LCD_WRITE, LCD_INSTR, 2, 1);	// 1 ms delay, 2
cycles
}

void LCD1602_backlight_off(void)
{
	PCF8574AT_send(0x0, 0, LCD_WRITE, LCD_INSTR, 2, 1);	// 1 ms delay, 2
cycles
}

void LCD1602_send_char(uint8_t character)
{
	PCF8574AT_send(character, 1, LCD_WRITE, LCD_DATA, 4, 3); // 4 cycles, 3 ms
delay
}

void LCD1602_init()
{
	// RS | R/!W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 - instruction
format from datasheet
	// DB7 | DB6 | DB5 | DB4 | LED | E | R/!W | RS    - I2C (PCF8547AT instruction
format)
```

```
	// softdelay(5);	// some (40 ms, for the best way) delay before
initialization start

	/*
	uint8_t instruction[] =
	{
		INTERFACE_8_BIT,
		INTERFACE_8_BIT,
		INTERFACE_4_BIT,
		INTERFACE_4_BIT_2_ROWS_FONT_5X8,
		DISPLAY_OFF_CURSOR_OFF_BLINK_DISABLE,
		DISPLAY_CLEAR,
		DIRECTION_L_TO_R_SHIFT_DISABLE,
		DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE
	};
	*/

	uint8_t instruction[] = {0x33, 0x32, 0x28, 0x0c, 0x01, 0x06, 0x80};

	//  - or this:
	//
0x33,0x32,INTERFACE_4_BIT_2_ROWS_FONT_5X8,DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE,
DISPLAY_CLEAR, DIRECTION_L_TO_R_SHIFT_DISABLE, DDRAM_ADDRESS_0X0
	// 0x30, 0x28, 0x28, 0xE, 0x1, 0x6 - my variant

	// uint8_t instr_count = sizeof(instruction) / sizeof(instruction[0]);
	// number of initialization instructions
	uint8_t instr_count = 7;

	uint8_t instr;

	for(instr = 0; instr < 3; instr++)
	{
		PCF8574AT_send(instruction[instr], 1, LCD_WRITE, LCD_INSTR, 2, 3);
	// 3 ms delay, 2 cycles
	}
	for(instr = 3; instr < instr_count; instr++)
	{
		PCF8574AT_send(instruction[instr], 1, LCD_WRITE, LCD_INSTR, 4, 3);
	// 3 ms delay, 4 cycles
	}
}

void LCD1602_set_cursor(uint8_t position)
{
	/*
		00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E
| 0F
		40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E
| 4F
	*/

	uint8_t data;

	data = (1 << 7) | position;

	PCF8574AT_send(data, 1, LCD_WRITE, LCD_INSTR, 4, 3);	// 4 cycles, 3 ms
delay
}

void LCD1602_send_char_position(uint8_t character, uint8_t position)
{
	LCD1602_set_cursor(position);
	LCD1602_send_char(character);
}
```

```
void LCD1602_position_rst(void)
{
	PCF8574AT_send(0x2, 1, LCD_WRITE, LCD_INSTR, 4, 1);	// 1 ms delay, 2
cycles
}

void LCD1602_display_clear(void)
{
	LCD1602_set_cursor(0x00);

	for(uint8_t position = 0x0; position <= 0x67; position++)
	{
		LCD1602_send_char(' ');
	}

	LCD1602_set_cursor(0x00);			// go to the start
}

void LCD1602_send_string(uint8_t *string, uint8_t position)
{
	LCD1602_set_cursor(position);

	while(*string)
	{
		LCD1602_send_char((uint8_t) *string);

		string++;
	}
}

void LCD1602_send_integer(uint16_t number, uint8_t position, uint8_t sign_num)
{
	if (sign_num > 0 && sign_num <= 5)
	{
		// uint16_t tens_of_thousands, thousands, hundreds, tens, units;
		uint8_t output[5];

		output[0] = number / 10000;
		output[1] = (number - (output[0] * 10000)) / 1000;
		output[2] = (number - (output[0] * 10000) - (output[1] * 1000)) / 100;
		output[3] = (number - (output[0] * 10000) - (output[1] * 1000) -
(output[2] * 100)) / 10;
		output[4] = (number - (output[0] * 10000) - (output[1] * 1000) -
(output[2] * 100) - (output[3] * 10));

		LCD1602_set_cursor(position);

		for(uint8_t i = (5 - sign_num); i < 5; i++)
		{
			LCD1602_send_char(output[i] + 48);
		}
	}
	else
	{
		LCD1602_send_string("?????", position);
	}
}
```