МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ

КУРСОВИЙ ПРОЕКТ

З курсу:

«Мікропроцесорні технології і компоненти РЕА»

тема: «Цифровий годинник з термометром»

Керівник:

доц. Корнєв В.П.

Допущено до захисту

“__”_____20__р.

Захищено з оцінкою

_____

Виконав:

Геращенко А. Ю.

студент IV курсу ФЕЛ

групи ДК-91

Київ – 2023

Національний Технічний Університет України

«Київський політехнічний інститут

імені Ігоря Сікорського »

Кафедра *Конструювання електронно-обчислювальної апаратури*

Дисципліна *Мікропроцесорні технології і компоненти РЕА*

Спеціальність *Інформаційно-обчислювальні засоби електронних систем*

Курс *IV* Група *ДК-91* Семестр *VII*

ЗАВДАННЯ

на курсовий проект студенту

Геращенко Артем Юрійович

(прізвище, ім'я, по батькові)

1. Тема проекту *Цифровий годинник з термометром*
2. Строк здачі студентом закінченого проекту (роботи) 16.01.2023
3. Вихідні дані до проекту (роботи)

   Спроектувати пристрій для контролю часу та температури, вихідні дані про час, температуру та вологість будуть виводитись на лсд дисплей, налаштування часу та дати буде здійснюватися кнопками керування.
4. Зміст розрахунково-пояснювальної записки (перелік питань, що розробляються):

   1. Опис структури пристрою і його складових

   2. Обґрунтування вибору елементної бази

   3. Опис і розрахунок схеми електричної принципової

   4. Алгоритм роботи програми

   5. Інструкція користувача

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

   1. Схема електрична принципова

   2. Перелік елементів

6. Дата видачі завдання 13.10.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапу курсового проекту (роботи)	Час виконання етапів проекту (роботи)	Примітка
1	*Створення схеми електричної принципової*	*14.10 – 2.11*	
2	*Опис структури пристрою і його окремих складових*	*4.11 – 12.11*	
3	*Обґрунтування 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*	
8	*Подача КП до захисту*	*20.12*	

Студент _____*Геращенко А.Ю.*

(підпис) (прізвище та ініціали)

Керівник_____*Корнєв В.П.*

(підпис) (прізвище та ініціали)

«__»_____2023 р.

ЗМІСТ

					ДК91.403272.001 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Геращенко А.Ю.			Цифровий годинник з термометром Пояснювальна записка	Літ.	Арк. 5 / Аркушів 71
Перевірив		Корнєв В.П.					
Н. Контр.						НТУУ «КПІ», ФЕЛ, гр. ДК-91	
Затвердив		Корнєв В.П.					

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

LCD	Liquid crystal display
RTC	Real-time clock
GPIO	General porpoise input-output
МК	Мікроконтролер
IC	Інтегральна схема
ШИМ	Широтно-імпульсна модуляція

ВСТУП

У повсякденному житті постійно постає необхідність моніторингу часу, вміння ефективно керувати часом зводить до мінімуму стрес і підвищує продуктивність — як в особистому, так і професійному плані. Правильний «тайм-менеджмент» не єдина складова продуктивності праці, для отримання максимального результату також необхідно відслідковувати значення вологості та температури оточуючого середовища. Висока температура повітря призводить до швидкої втоми, перегрівання організму і теплового удару. Крім того, висока температура повітря порушує водносольовий обмін в організмі. Низька температура і великі швидкості руху повітря при тривалому впливі призводять до розладу кровообігу, сприяють захворюванню на ревматизм, грип і хвороби дихальних шляхів. Низька вологість повітря сушить шкіру, слизові оболонки. У горлі, носі, очах може з'являтися сухість та неприємні відчуття. Людина стає вразливою для вірусів та бактерій. Висока вологість підвищує віддачу тепла від тіла людини. Самопочуття погіршується, з'являється слабкість. Надлишок вологи може спричинити загострення серцево-судинних захворювань. Окрім впливу на людину, температура та вологість впливають на стан речей, які знаходяться в робочому приміщенні.

Наразі на ринку представлена велика кількість годинників, які мають додатковий функціонал, окрім мінімально необхідних виведення часу та дати. Зокрема наведено моделі, які дають змогу відслідковувати температуру та вологість у приміщенні.

В рамках даної роботи розглядатиметься розробка цифрового годинника з додатковими функціями вимірювання температури та вологості. Оскільки сьогодні сімейство ARM займає приблизно 75 % всіх портативних 32-бітних процесорів і вони є найбільш використовуваними за рахунок їх доступності за ціною та великою кількістю документації на них, то у проекті прийнято використовувати мікроконтролер даного сімейства від компанії STMicroelectronics – STM32F407. Адже в даному МК реалізовані всі необхідні нам функції, серед яких: зовнішні переривання, RTC, доволі висока частота

тактового сигналу, і він має відносно невелику вартість. Отримані дані часу, дати, температури та вологості будуть відображатися на знакосимвольному дисплеї, налаштування часу та дати здійснюється через тактові кнопки.

РОЗДІЛ 1. ОПИС СТРУКТУРИ ПРИСТРОЮ І ЙОГО ОКРЕМИХ СКЛАДОВИХ

1.1 Структура пристрою

Структурна схема пристрою наведена на рисунку 1.1.

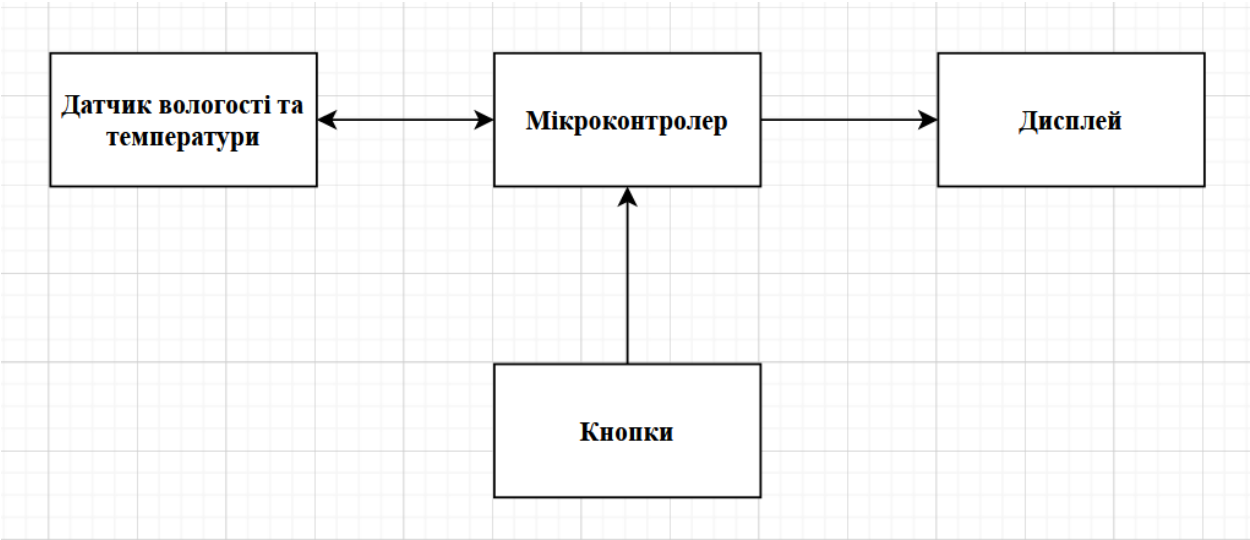

Рисунок 1.1 – Структурна схема пристрою

Пристрій складається з чотирьох логічних блоків. Перший логічний блок дає змогу вимірювати вологість та температуру і надавати виміряні дані до мікроконтролера. Другий логічний блок приймає інформацію від користувача шляхом натискання на кнопки, і передає цю інформацію мікроконтролеру у вигляді сигналів. Третій логічний блок являє собою мікроконтролер STM32F407VG, який здійснює обробку отриманої від попередніх двох логічних блоків інформації та передає її до блоку виводу даних. Четвертий логічний блок – це блок виведення даних, до цього блоку належить символьний LCD дисплей, який надає користувачу інформацію про поточний час та дату і оброблену інформацію вимірів датчика температури та вологості.

1.2 Принцип вимірювання температури та вологості

У даному приладі використовується цифровий датчик температури та вологості DHT11 (рисунок 1.2).

Рисунок 1.2 – Датчик температури та вологості

DHT11 складається з ємнісного датчика вологості, термістора для вимірювання температури та IC. Конденсатор містить два електроди з вологоутримуючою підкладкою-діелектриком між ними. Зміна рівня ємності відбувається за зміни рівня вологості. Використаний у датчику термістор має від'ємний температурний коефіцієнт, який викликає зменшення значення опору при підвищенні температури. IC вимірює, обробляє зміну значень ємності та опору і перетворює їх у цифрову форму.

Основні характеристики датчика[1]:

- Напруга живлення від 3В до 5В;
- Максимальне споживання струму 2,5 мА (під час запиту даних):
- Частота дискретизації не більше 1 Гц.
- Діапазон вимірювань температури 0 - 50°С (точність ± 2°С);
- Діапазон вимірювань вологості 5 – 95% (точність ± 5%)
- Розмір 15,5 х 12 х 5,5 мм;

Схема підключення до МК [1] реалізована через один провідник, тобто відбувається обмін даними за протоколом 1-Wire (рисунок 1.3).

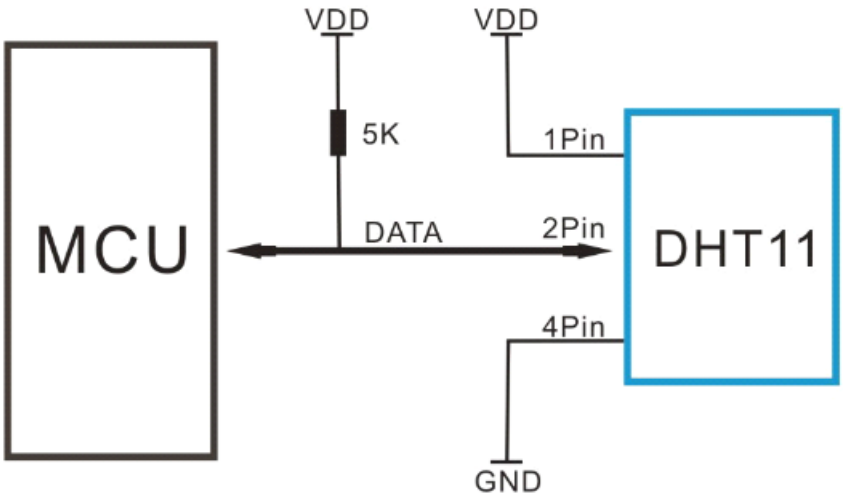

Рисунок 1.3 – Типова схема підключення датчика до МК

1.3 Принцип виводу інформації

В якості засобу відображення інформації в даному приладі використовується LCD дисплей (рисунок 1.4) типу 1602 на основі контролеру HD44780, який містить два рядки по 16 символів в кожному.

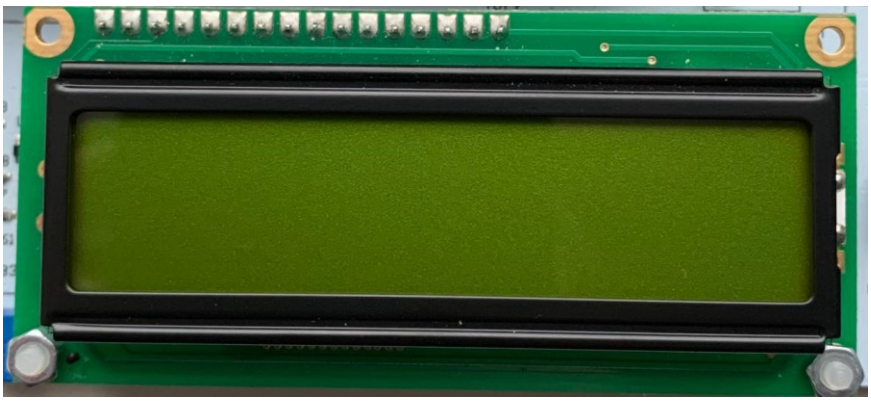

Рисунок 1.4 – LCD дисплей WH1602B

Даний дисплей має вісім ліній даних, кожна лінія призначена для передачі одного біту даних. Також передбачено вибір між 8-ми та 4-ох бітною схемою підключення. Для першого варіанту інформація буде передаватись швидше, але необхідно буде задіяти 8 виводів МК. Для другого варіанту необхідно лише 4 виводи МК, даний варіант реалізує передачу даних півбайтами і, як наслідок швидкість передачі зменшується, але враховуючи відсутність потреби швидкого відображення інформації на дисплей, для реалізації приладу буде використовуватись саме цей варіант підключення, бо він дозволяє зекономити 4 виводи МК. Крім інформаційних ліній дисплей два

виводи для під’єднання живлення і один вивід для керування контрастністю. Характеристики дисплею [2]:

- Розмір 80 х 36 мм
- Робоча температура 0 - 50℃
- Формат 16 х 2
- Розмір символа 5.56 х 2.96 мм
- Розмір крапки 0.5 х 0.5 мм
- Інтерфейс HD44780
- Видима область 66.0 х 16.0 мм
- Живлення 5В

Оснащення дисплею мікросхемою HD44780 робить його «розумнішим», збільшує кількість команд керування.

1.4 Принцип введення даних

Для конфігурації роботи пристрою використовується п’ять тактових кнопок (рисунок 1.5).

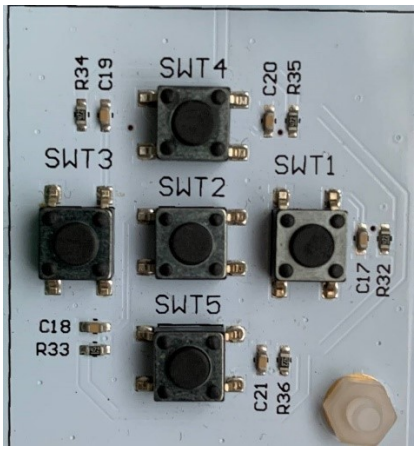

Рисунок 1.5 – Кнопки керування

Кнопки SWT4, SWT5 (рисунок 1.5) відповідають за налаштування часу, годин та хвилин відповідно. Кнопки SWT3, SWT2, SWT1 відповідають за налаштування дати у форматі день/місяць/рік відповідно. Налаштування відбувається внаслідок інкрементування даних на 1 при кожному натисканні, також встановлено умови переповнення за яких значення відповідного розряду часу, або дати скидається та інкрементація починається спочатку.

РОЗДІЛ 2. РОЗРОБКА СХЕМИ ЕЛЕКТРИЧНОЇ ПРИНЦИПОВОЇ

2.1 Вибір елементної бази

2.1.1 Вибір мікроконтролера

У даному пристрої було використано МК серії STM32F407VGT у корпусі LQFP100[3] (рисунок 2.1).

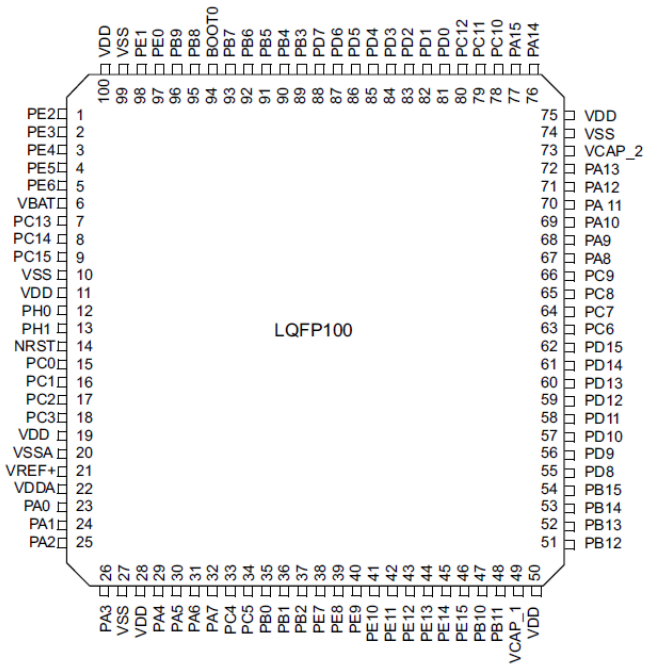

Рисунок 2.1 – Кнопки керування

Серед причин вибору даного мікроконтролеру були наступні:

- Популярність і відносно дешева ціна;
- Можливість обробляти до 16 зовнішніх переривань, що задовольняє кількості кнопок управління;
- Наявність вбудованого модуля RTC, який дозволяє відслідковувати реальний час та дату;
- Висока продуктивність роботи при невисокому енергоспоживанні;
- Наявність документації у вільному доступі, яку надає виробник;
- 16 портів GPIO, перші три з яких, A, B та C, мають до 16 виводів, чого цілком достатньо для підключення LCD дисплею, кнопок та датчика.

Зважаючи на наведені причини, можна вважати вибір даного мікроконтролеру цілком аргументованим у рамках даного проекту.

2.1.2 Вибір датчика температури та вологості

У якості датчика температури та вологості у даному пристрої буде використовуватись цифровий датчик DHT11. Даний вибір обґрунтований тим, що датчик використовує протокол 1-Wire для обміну інформацією, який потребує лише одного виводу для підключення до МК. Також під час вибору датчика було відмічено, що він є цифровим і вихідний сигнал не потребує обробки завдяки АЦП, дані особливості суттєво спрощують комунікацію датчика з МК. Наступним фактором є компактні розміри, невелика ціна та популярність використання датчику для побудови метеостанцій, що свідчить про його якість та прийнятну точність.

2.1.3 Вибір кнопок

Серед компонентів блоку введення даних наявні кнопки у кількості п’яти штук. Особливий вимог до вибору кнопок немає, тому було використано кнопки, наявні на платі розширенні Global logic. Дані кнопки мають невеликі розміри та низьку ціну, що робить їх цілком прийнятними для використання в рамках реалізації приладу.

2.1.4 Вибір дисплею

У даному проекти використовується LCD дисплей WH1602B. Вибір даного дисплею було обумовлено наступними чинниками:

- Наявність у складі дисплею мікросхеми HD44780, яка суттєво розширює його функціонал.
- Низька потужність споживання, в порівнянні з іншими типами дисплеїв.
- Вбудована таблиця символів, що забезпечує легкість роботи з дисплеєм.
- Можливість підключення «напряму» до виводів МК.
- Невеликі габаритні розміри та достатня кількість місця для виводу дати, часу, температури та вологості.

2.2 Підключення елементів до мікроконтролера

2.2.1 Підключення датчику температури

Як вже згадувалось у попередньому розділі, датчик DHT11 підключається до МК за допомогою інтерфейсу 1-Wire, схема підключення [4] наведена на рисунку 2.2.

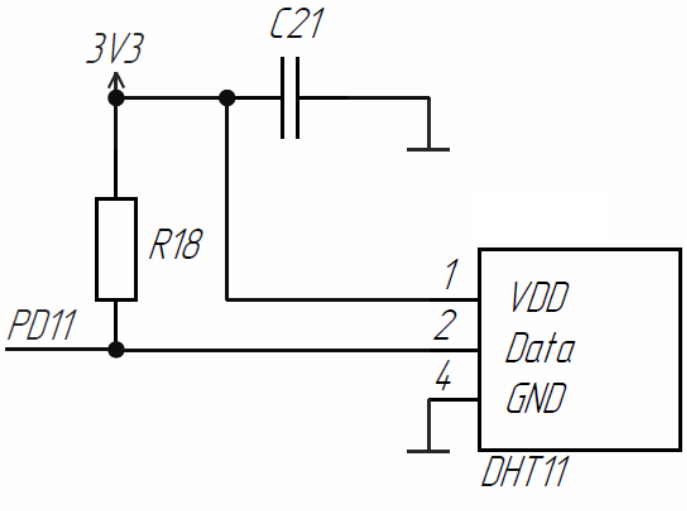

Рисунок 2.2 – Підключення датчику DHT11

Як видно на рисунку 2.2 лінія даних датчика завжди має бути «підтянута» через резистор до лінії живлення. Також важливим є правильна конфігурація виводу МК, який під'єднано до датчика, його необхідно конфігурувати, як вихід з відкритим стоком для того, щоб датчик міг встановлювати на лінії високий та низький рівні та передавати інформацію. Наведені вище налаштування встановлені стандартом 1-Wire і є обов'язковими для коректної роботи датчика.

Для отримання вимірів від датчика необхідно дотримуватись наступного алгоритму:

1. МК виконує звернення до датчика[5] (рисунок 2.3) встановлюючи низький рівень на виводі до якого підключено провідник Data (рисунок 2.2) не менше ніж на 18 мкс, потім встановлює високий рівень на 20-40 мкс. Після звільнення лінії даних вивід налаштовується як input для отримання даних від датчика.

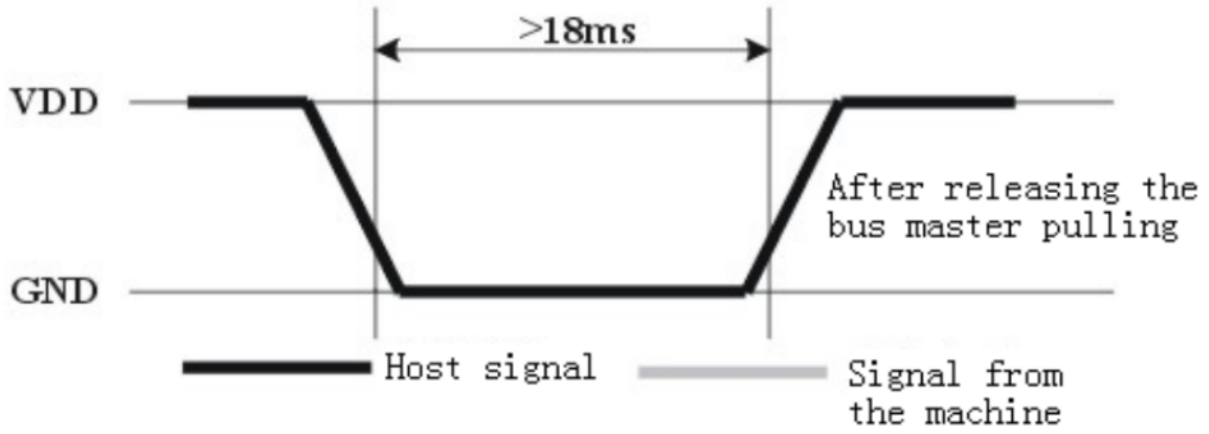

Рисунок 2.3 – Стартовий сигнал звернення МК до датчика

2. Датчик відповідає МК встановлюючи низький рівень на виводі даних на 80 мкс і високий рівень на 80 мкс[5] (рисунок 2.4). Дана команда від датчика інформує мікроконтролер про готовність відправки вимірів.

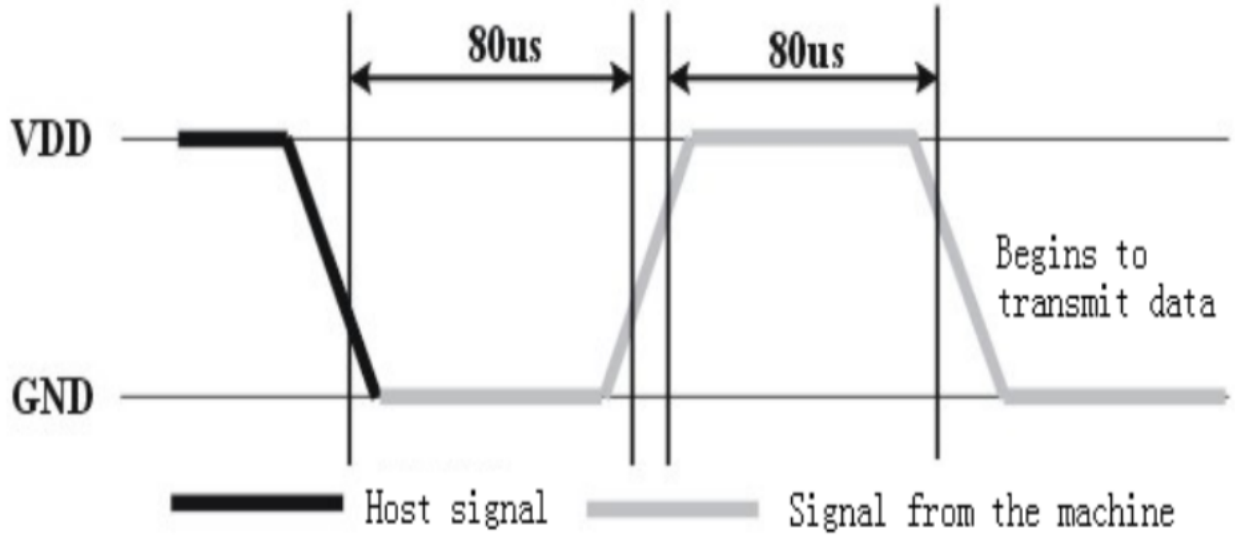

Рисунок 2.4 – Сигнал готовності датчика до передачі вимірів

3. Датчик передає дані до МК, довжина даних – 40 біт, тобто 5 пакетів по 8 біт. Перший та другий пакети з 8 біт кожний, містять в собі цілу та дробову частину значення вологості відповідно. Третій та четвертий пакети з 8 біт кожний, містять в собі цілу та дробову частину значення температури відповідно. Останній, п’ятий пакет з 8 біт містить в собі контрольну суму, яка буде використовуватись для перевірки цілісності отриманих даних під час написання програми. Кожний біт даних починається з сигналу низького рівня протягом 50 мкс, період часу сигналу високого рівня визначає чому буде дорівнювати біт даних, «0»

чи «1». Якщо тривалість високого рівня триває у межах 26-28 мкс – цей біт даних буде записано, як «0». У разі тривалості високого рівня сигналу 70 мкс – біт даних буде записано, як «1». Після передачі всіх 40 біт даних лінія передачі встановлюється в низький рівень на 50 мкс і звільняється, «підтягуючий» резистор встановлює на лінії високий рівень, вказуючи на її звільнення[5] (рисунок 2.5).

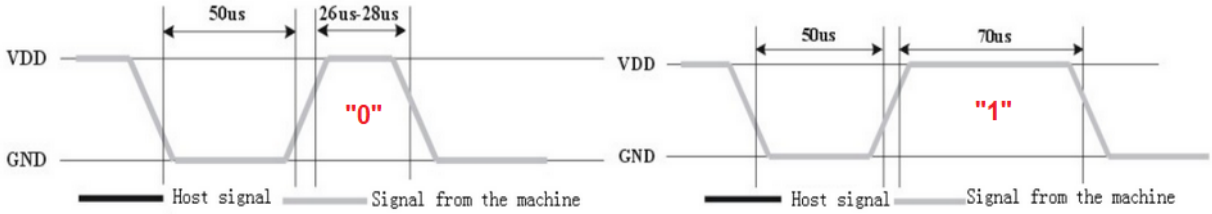

Рисунок 2.5 – Форма сигналу передачі «0» та «1»

2.2.2 Підключення дисплею

Дисплей LCD WH1602B підключено до МК через 4-х бітний інтерфейс[4], тобто для взаємодії з дисплеєм буде використовуватись лише 4 останні біти D4, D5, D6, D7 (рисунок 2.6), тому дані будуть передаватись півбайтами за два такти.

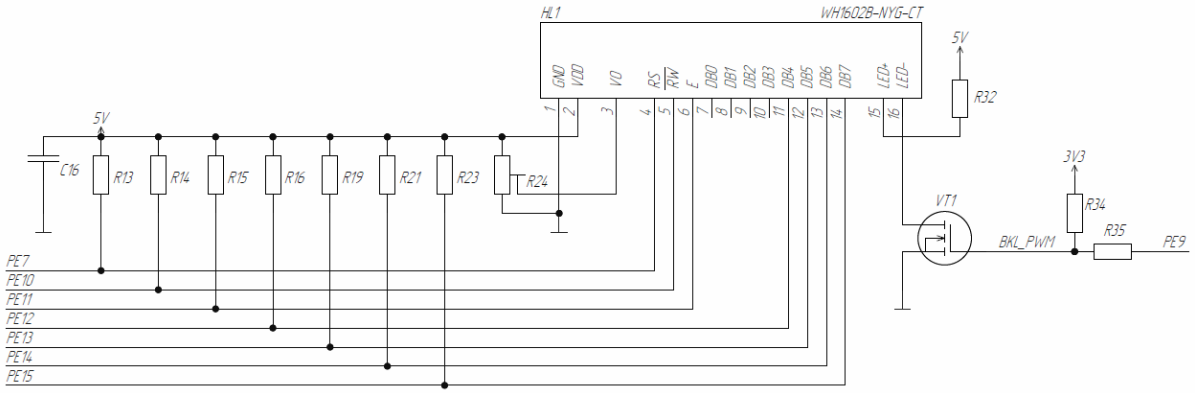

Рисунок 2.6 – Підключення дисплею до МК

Даний дисплей має наступні контакти:

- Vss: «-» живлення дисплею;
- VDD: «+» живлення дисплею;
- VO: управління контрастністю екрану;
- RS: вибір регістру;
- RW: вибір режиму запис/читання;

- E: вхід дозволу;
- D0 – D7: виводи даних;
- A: «+» живлення підсвітки;
- K: «-» живлення підсвітки.

Даний дисплей дозволяє як зчитування, так і запис даних, але у рамках даного проекту використано лише функцію запису.

На схемі підключення зображено 7 резисторів номіналом 4,7 кОм, дані резистори використовують «підтяжку» ліній даних до 5В, це необхідно для реалізації 5В логіки адже МК працює з напругою 3,3В, а дисплей має номінальну напругу 5В. Для надання необхідного струму рекомендовано перевести виводи МК, до яких підключено дисплей у режим open drain. Змінний резистор, який підключено до виводу VO дозволяє керувати контрастністю символів на дисплеї. Транзистор VT1 підключено в ключовому режимі, тому подаючи на вивід PE9 ШИМ сигнал можна регулювати яскравість дисплею.

Для керування дисплеєм застосовуються наступні рішення: встановлюються сигнали RS та R/W, сигнал RS показує те, що саме буде передаватись по лініях DB0 – DB7, RS рівний «0» вказує на передачу інструкції, а RS рівний «1» вказує на передачу даних. Далі сигнал R/W встановлюється в «0» - дані передаються від МК до дисплею.

Після встановлення відповідних станів на виходах RS та R/W необхідно очікувати певний проміжок часу. Далі сигнал на виході E встановлюється у «1» - можна продовжувати запис даних. Часові діаграми для запису даних [6]. Часові діаграми для запису даних вказано на рисунку 2.7.

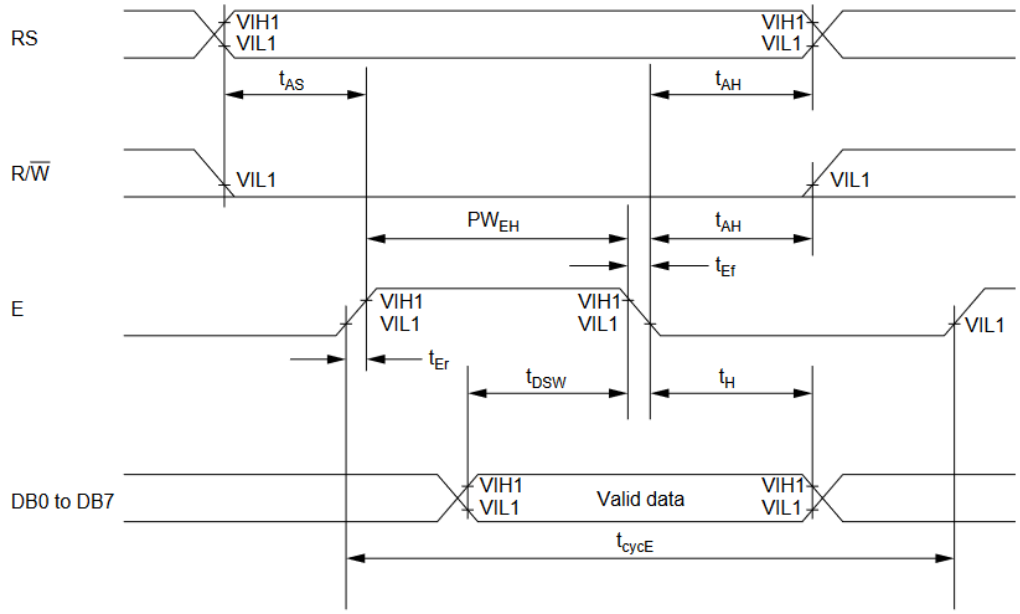

Рисунок 2.7 – Часова діаграма запису даних в дисплей

2.2.3 Підключення кнопок

Для реалізації пристрою використано п’ять кнопок, які «підтянуто» до живлення через резистори 4,7 кОм для формування високого логічного рівня, коли кнопка не натиснута та низького логічного рівня під час натискання кнопки. Також до кожної кнопки підключено конденсатори з малою ємністю, таке підключення конденсаторів реалізує боротьбу з брязкотом контактів. Схема підключення кнопок[4] зображена на рисунку 2.8.

На схемі можна побачити, що кнопка SB3 підключена до виводу PC11 мікроконтролера, кнопка SB4 під’єднана до виводу PA15, кнопка SB5 під’єднана до виводу PC9, кнопка SB6 під’єднана до виводу PC6, кнопка SB7 під’єднана до виводу PC8.

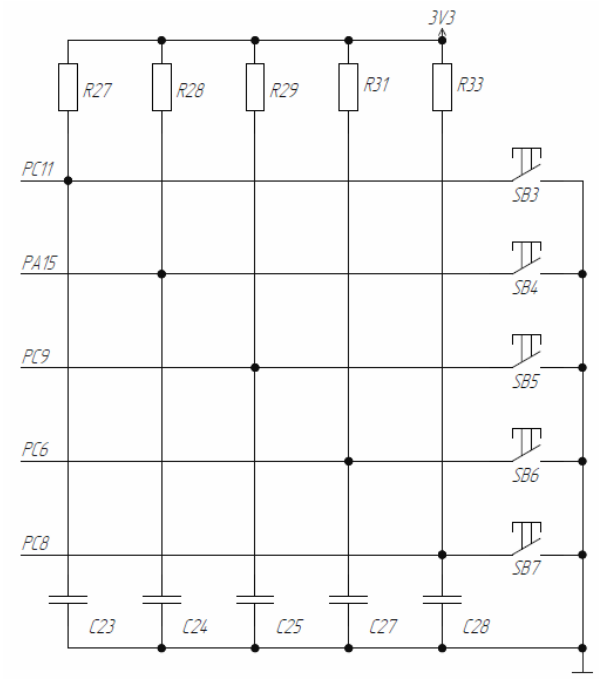

Рисунок 2.8 – Підключення кнопок

2.2.4 Інші підключення до мікроконтролера

Для правильної роботи мікроконтролера та для полегшення відладки програми до мікроконтролера під’єднано наступну обв’язку:

- Кнопка скидання – необхідна для скидання поточного стану виконання програми у початок. Як видно з рисунку 2.9 [7] кнопка «підтянута» до напруги живлення, що означає про наявність на виводі NRST високого рівня, коли кнопка не натиснута. Також схема з’єднання містить конденсатор для захисту від брязкоту контактів.

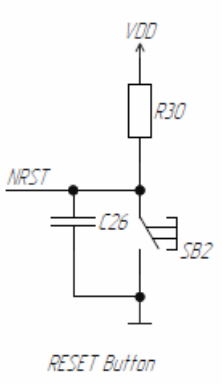

Рисунок 2.10 – Підключення кнопки скидання

- Кварцові резонатори [7] ZQ1 (рисунок 2.10) та ZQ2 (рисунок 2.11) необхідні для отримання тактового сигналу схеми, завдяки ним

працюють усі фунції МК (наприклад таймери, RTC), які передбачають використання тактової частоти.

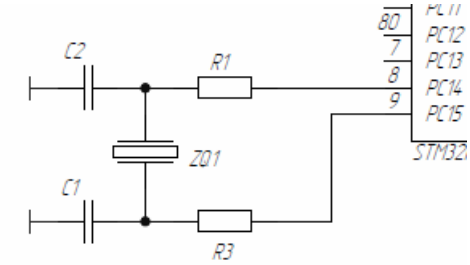

Рисунок 2.10 – Підключення кварцового резонатора ZQ1 з обв’язкою

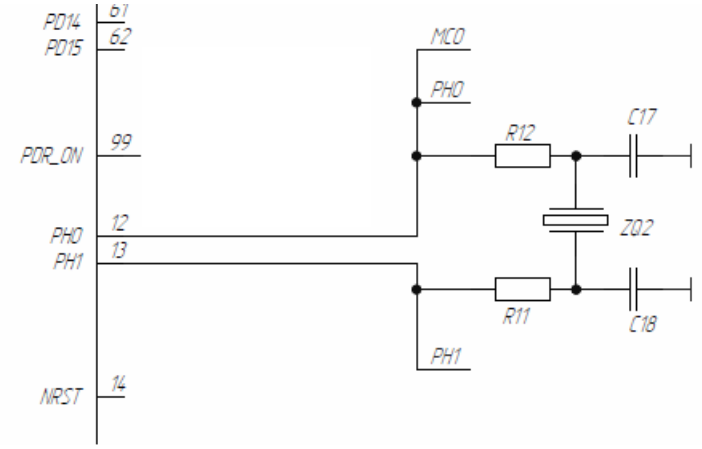

Рисунок 2.11 – Підключення кварцового резонатора ZQ2 з обв’язкою

- Схема для стабілізації напруги живлення мікросхеми зображена на рисунку[7] 2.12.

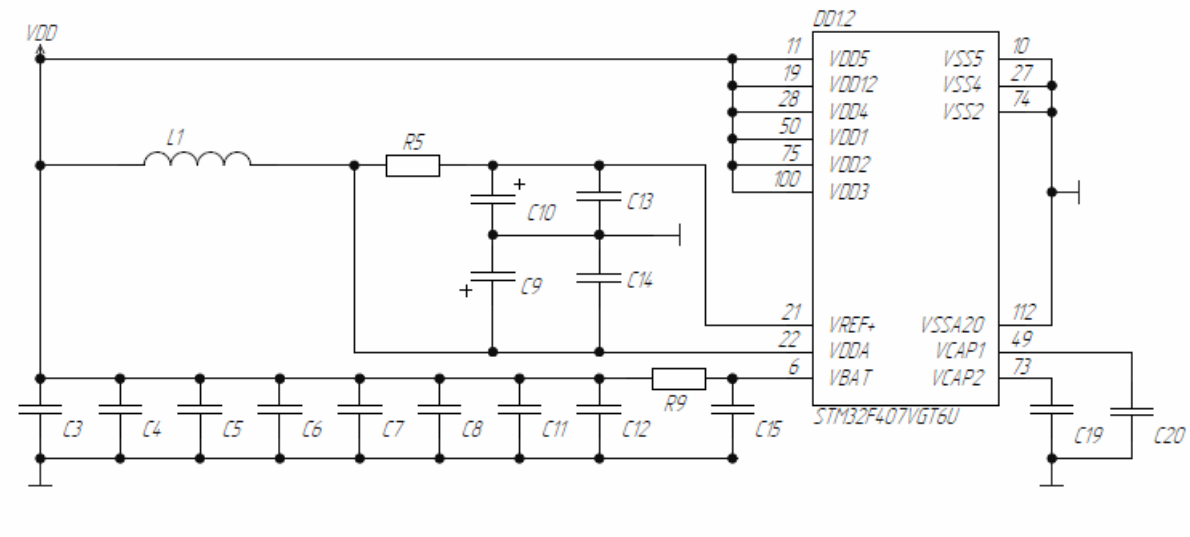

Рисунок 2.11 – Підключення стабілізації напруги живлення

2.3 Принцип роботи схеми електричної принципової

Схема електрична принципова наведена у додатку на кресленні ДК91.303272.001Е3.

Схема складається з мікроконтролера до якого підключено наступну периферійні модулі: датчик температури та вологості, LCD дисплей, п’ять тактових кнопок.

На схемі зображено підключення датчика DHT11 до виводу GPIOD PD11 мікроконтролера, для коректної роботи вивід мікроконтролера, який підключено до лінії даних, необхідно встановити в режим відкритого стоку. Оскільки датчик використовує протокол підключення 1-Wire необхідно робити «підтяжку» лінії даних до живлення, що і зроблено на схемі підключення. Також для фільтрування пульсацій схема підключення містить конденсатор. Комунікація з датчиком забезпечується завдяки подачі сигналів від мікроконтролера з витримкою затримок, які зазначено в документації [1] та описано раніше, для реалізації затримки в мікросекундах використано таймер TIM1.

Для підключення LCD дисплея до МК використано 7 портів GPIOE, вивід RS підключено до порту PE7, вивід R/W підключено до порту PE10, вивід E підключено до порту PE11 (опис призначення портів наведено у розділі 2). Для передачі даних використовується 4 бітний режим, передача здійснюється півбайтами за два такти через виводи D4-D7, які підключено до портів PE12-PE14 відповідно. Оскільки МК працює з 3,3В логікою, а дисплей підключається до напруги живлення 5В зроблено схему узгодження шляхом підтяжки виводів даних та RS, R/W, E до 5В через резистори 4,7 кОм. Щоб забезпечити коректні роботу даної схеми підключення необхідно встановити виводи МК, які підключено до LCD дисплею, у режим open drain.

Схема підключення містить змінний резистор, який підключено до виводу VO, завдяки цьому резистору можна регулювати різкість зображення на дисплеї.

Виводи LED + та LED – використовуються для ввімкнення підсвітки дисплею, до виводу LED – підключено транзистор, який працює у режимі ключа, він може використовуватись для зміни яскравості дисплея. Якщо на вивід PE9 подавати ШИМ сигнал, можна змінювати яскравість.

Тактові кнопки під’єднано до портів GPIOA та GPIOC, кнопки SB3, SB5, SB6, SB7 підключені до виводів PC11, PC9, PC6, PC8 відповідно. Кнопку SB4 під’єднано до виводу PA15. На схемі підключення зображено «підтяжку» кнопок до напруги живлення, тому при натисканні кнопок на виводах буде напруга логічного «0». Також реалізовано захист від брязкоту контактів завдяки підключенню конденсаторів номіналом 0,01 мкФ.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис ресурсів мікроконтролера та принцип взаємодії з ними

3.1.1 Використання GPIO

GPIO є користувацькими портами вводу-виводу, які в залежності від потрем можна по різному конфігурувати. Структура піна GPIO [8] наведена на рисунку 3.1.

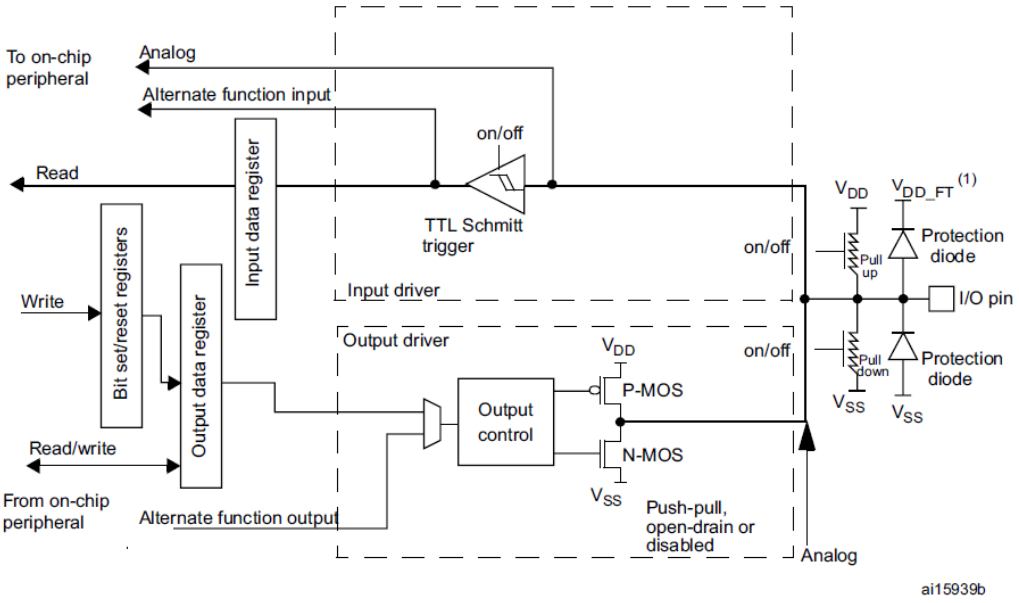

Рисунок 3.1 – Базова структура піна порта вводу-виводу

У даному проекті через порти GPIO до МК під’єднано таку периферію: кнопки керування, датчик температури, дисплей. Для підключення кожного периферійного модуля було налаштовано порти вводу-виводу. Найменше налаштувань було задіяно для портів до яких під’єднано кнопки та дисплей тому, що сигнали через ці піни передаються лише від МК, або до МК. Для налаштування виводу до якого підключено датчик було задіяно більше налаштувань, оскільки, сигнали по лінії даних передаються в обидві сторони.

Для роботи з портами GPIO, перш за все, потрібно пдати тактову частоту, для цього необхідно встановити «1» у відповідний біт регістру RCC_AHB1ENR [8], регістр зображено на рисунку 3.2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Res.	**DMA2DEN**	DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.1 – Структура регістру RCC_AHB1ENR

Датчик, дисплей та кнопки використовують порти GPIOA, GPIOC, GPIOD, GPIOE.

Після подачі тактування на порти, які використано для реалізації пристрою, необхідно виконати налаштування режиму роботи. Для налаштування режиму роботи використовується регістр [8] GPIOx_MODER (рисунок 3.1).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.2 – Структура регістру GPIOx_MODER

Встановлюючи різні комбінації бітів у комірки даного регістру можна налаштовувати режим роботи відповідного виводу. Передбачені наступні режими роботи:

- Input, необхідно записати у комірку регістру 00;
- Output, необхідно записати у комірку регістру 01;
- Alternate function, необхідно записати у комірку регістру 10;
- Analog, необхідно записати у комірку регістру 11;

В рамках даного проекту використано лише два режими роботи: output та input.

Після визначення необхідного режиму роботи та його налаштування необхідно задати тип виходу для пінів, які було налаштовано як output.

Налаштування типу виходу піна виконуються в регістрі GPIOx_OTYPER [8] (рисунок 3.3).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.2 – Структура регістру GPIOx_OTYPER

Якщо записати «1» у відповідний біт, то відповідний вихід МК конфігурується як вихід з відкритим стоком (open drain) – саме це налаштування використовується для підключення датчика DHT11 та LCD дисплею. У разі ігнорування налаштування типу виходу робота з датчиком температури та вологості буде неможлива, це принциповий момент, адже датчик повинен мати змогу керувати рівнями сигналу на лінії даних, чого не може бути виконано при налаштуванні за замовчуванням, коли МК жорстко тримає встановлений рівень.

Досить важливим при конфігурації пінів є регістр GPIOx_PUPDR [8] (рисунок 3.3), який відповідає за встановлення «підтяжки» виводу до землі чи живлення.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.2 – Структура регістру GPIOx_PUPDR

Даний регістр було задіяно для налаштування піна PD11, який під’єднано до датчика та пінів до яких підключено LCD диспей і встановлено налаштування «00» у відповідних комірках регістру, тобто налаштовано режим «без підтяжки». Для реалізації логіки роботи кнопок було налаштовано «підтяжку» до землі «10» – при натисканні кнопки на виводі буде високий логічний рівень «1».

Після налаштування портів вводу-виводу можна використовувати регістри [8] GPIOx_BSRR (рисунок 3.3) та GPIOx_IDR (рисунок 3.4).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Рисунок 3.3 – Структура регістру GPIOx_BSSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Рисунок 3.4 – Структура регістру GPIOx_IDR

Регістр GPIOx_BSSR складається з двох частин: старші 16 біт відповідають за встановлення низького рівня на виходу, а молодші – за встановлення високого рівня. Щоб налаштувати рівень необхідно встановити «1» у відповідну комірку регістру, встановлення «0» ні до чого не призводить. Регістр GPIOx_IDR використовується для відслідковування станів сигналів на виводах. Для зчитування доступні 16 молодших біт регістру, запис у комірки цього регістру є неможливим.

3.1.2 Використання таймера TIM1

Для реалізації протоколів обміну даними між МК і датчиком DHT11, між МК та LCD дисплеєм необхідно витримувати зазначені у документації затримки. На основі таймера TIM1 реалізовано функцію, яка створює затримку у мікросекундах. Загалом даний мікроконтролер має чотири види таймерів:

- Системний таймер SysTick;
- Базові таймери TIM6 та TIM7;
- Таймери загального призначення TIM2-TIM5 та TIM9-TIM14;
- Складні таймери TIM1 та TIM8;

В залежності від складності таймера, він здатен виконувати різні задачі, наприклад складні таймери здатні генерувати ШИМ сигнали на виході МК, здатні вимірювати довжину імпульсу, який подається на вхід МК.

Також таймери можуть здійснювати лічбу вгору і вниз, підтримують управління електроприводом.

Таймери загального призначення можуть виконувати ті ж задачі, що і складні таймери, але вони не підтримують управління електроприводом.

Метою використання таймера у даному проекті є створення функції, яка здатна реалізувати затримку в мікросекундах, для подібних цілей потрібен досить точний таймер. Для виконання поставленої задачі підходить будь-який таймер зі списку доступних, крім системного. Оскільки реалізація проекту не потребує використання великої кількості таймерів, прийнято рішення використовувати TIM1.

Дозвіл на тактування таймера можна задати у регістрі RCC_APB2ENR [8] (рисунок 3.5).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					LTDC EN	Reserved			SAI1EN	SPI6EN	SPI5EN	Res.	TIM11 EN	TIM10 EN	TIM9 EN
					rw				rw	rw	rw		rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reser-ved	SYSCFG EN	SPI4EN	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reserved		USART 6 EN	USART 1 EN	Reserved		TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw			rw	rw			rw	rw

Рисунок 3.5 – Структура регістру RCC_APB2ENR

Для встановлення дозволу тактування необхідно встановити 0-й біт регістру в «1».

Тактова частота може бути зменшена за допомогою використання подільника PSC, значення подільника частоти встановлюється в регістрі [8] TIM1_PSC (рисунок 3.6).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.6 – Структура регістру TIM1_PSC

Значення лічильника зберігається в регістрі [8] TIM1_CNT (рисунок 3.7).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.7 – Структура регістру TIM1_CNT

Для налаштування таймеру також використовується регістр контролю таймера[8] TIM1_CR1 (рисунок 3.8). У даному регістрі знаходиться біт дозволу відліку таймера, запис в нього «1» починає відлік таймера, запис «0» забороняє відлік.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.8 – Структура регістру TIM1_CR1

Регістр [8] TIM1_ARR (рисунок 3.9) зберігає значення при якому таймер автоматично перезапуститься, після чого його значення CNT буде рівним 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.9 – Структура регістру TIM1_ARR

Також при налаштуванні таймеру використовується регістр [8] TIM1_SR (рисунок 3.10). У даному регістрі наявний лише один біт зчитування, а саме UIF. Даний прапор встановлюється тоді, коли завершено налаштування таймера, а саме оновлення вмісту всіх його регістрів. Після встановлення цього прапора в «1» можна починати працювати з таймером.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0	Res.	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Рисунок 3.10 – Структура регістру TIM1_SR

3.1.3 Використання зовнішніх переривань

У даному проекті для налаштування значень дати та часу використовується п’ять тактових кнопок. При натисканні на кожну з них викликається зовнішнє переривання і виконується функція-обробник переривання, яка в залежності від натиснутої кнопки інкрементує відповідні поля часу або дати.

Для налаштування переривань необхідно подати тактову частоту на контролер системних конфігурацій SYSCFGC, який виконує налаштування контролеру переривань та оброблює зовнішні переривання. Подати тактування можна записавши у регістр RCC_APB2ENR (рисунок 3.5) значення «1» у відповідний біт.

Після цього викнують налаштування зовнішніх переривань. Спочатку необхідно вказати лінію до якої підключено джерело сигналу, яке викликатиме зовнішнє переривання. Всього існує 23 зовнішніх переривання, 16 з яких відведено для переривань портів GPIO, інші для переривань від RTC, USC, Ethernet. До кожної з 16 ліній підключено 16 пінів номера яких однакові з номерами портів GPIO, всі 16 пінів підключені через мультиплексор, який обирає пін прийому сгналу. Записуючи відповідне значення у тетради EXTIx_[3:0] регістру [8]SYSCFG (рисунок 3.11) можна обирати джерело переривань.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.11 – Структура регістру SYSCFG

Для того, щоб МК реагував на запит переривання необхідно зняти маску переривання, зробити це можна у регістрі [8] EXTI_IMR (рисунок 3.12).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.12 – Структура регістру EXTI_IMR

Встановлення «1» у відповідний біт регістру знімає маску з переривання і МК починає на нього реагувати.

Також при налаштуванні переривань необхідно обрати на який фронт сигналу, передній чи задній, буде реагувати переривання.

Для заданих цілей використовують регістри [8] EXTI_RTSR (рисунок 3.13) та EXTI_FTSR (рисунок 3.14). Запис «1» у відповідний біт налаштовує фронт під час якого буде виконуватись переривання, слід вказати, що регістри є незалежними один від одного і, тому, можна налаштувати спрацювання переривання по передньому і задньому фронтам.

Останнім кроком може бути виконано налаштування пріоритетності виконання переривань.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.13 – Структура регістру EXTI_RTSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.14 – Структура регістру EXTI_FTSR

3.1.4 Використання RTC

Годинник реального часу (RTC) — це незалежний таймер/лічильник BCD. RTC забезпечує годинник/календар часу доби, два програмованих переривання будильника та періодичний програмований прапор пробудження з можливістю переривання. RTC також містить блок автоматичного пробудження керувати режимами низького енергоспоживання. Два 32-розрядних регістри містять секунди, хвилини, години (12- або 24-годинний формат), день (день тижня), дату (день місяця), місяць і рік, виражені в двійковому десятковому форматі. (BCD). Значення субсекунд також доступне в двійковому форматі. Компенсації за 28-, 29- (високосний рік), 30- та 31-денні місяці здійснюються автоматично. Також можна виконати компенсацію літнього часу.

Додаткові 32-розрядні регістри містять програмовані субсекунди, секунди, хвилини, години, день і дату. Функція цифрового калібрування доступна для компенсації будь-яких відхилень у точності кристалічного генератора. Після скидання резервного домену всі регістри RTC захищені від можливого паразитного запису доступу.

Поки напруга живлення залишається в робочому діапазоні, RTC ніколи не зупиняється, незалежно від стану пристрою (режим роботи, режим низького енергоспоживання або скидання). Для налаштування RTC у даному проекті використано наступні регістри:

- RTC_TR — це тіньовий регістр календарного часу. Цей регістр повинен бути записаний лише в режимі ініціалізації [8] (рисунок 3.15), він надає змогу обрати формат часу 24 годинний або 12 годинний та використовується для отримання значень поточного часу, які зберігаються у відповідних полях.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PM	HT[1:0]		HU[3:0]			
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Рисунок 3.15 – Структура регістру RTC_TR

- RTC_DR — це тіньовий регістр календарної дати. Цей регістр повинен бути записаний лише в режимі ініціалізації [8] (рисунок 3.16). Регістр надає можливість прочитати поточне значення дати.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								YT[3:0]				YU[3:0]			
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]			MT	MU[3:0]				Reserved		DT[1:0]		DU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Рисунок 3.16 – Структура регістру RTC_DR

- RTC_CR – це регістр, який містить у собі багато налаштувань для калібрування роботи RTC [8] (рисунок 3.17).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								COE	OSEL[1:0]		POL	COSEL	BKP	SUB1H	ADD1H
								rw	rw	rw	rw	rw	rw	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	DCE	FMT	BYPS HAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.17 – Структура регістру RTC_CR

- RTC_ISR – це регістр, який використовують для ініціалізації RTC [8] (рисунок 3.18).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															RECALPF
															r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TAMP 2F	TAMP 1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUT WF	ALRB WF	ALRA WF
	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rw	r	rc_w0	r	r	r	r	r

Рисунок 3.18 – Структура регістру RTC_ISR

- RTC_WPR – це регістр у який необхідно записати ключ для отримання дозволу запису (у даному проекті зміни значень часу та дати) [8] (рисунок 3.19). Щоб розблокувати захист від запису для всіх регістрів RTC, крім RTC_ISR[13:8], RTC_TAFCR і RTC_BKPxR, необхідно виконати такі дії: запишіть «0xCA» в реєстр RTC_WPR, запишіть «0x53» у регістр RTC_WPR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								KEY							
								w	w	w	w	w	w	w	w

Рисунок 3.19 – Структура регістру RTC_WPR

3.2 Налаштування ресурсів мікроконтролера в STM32CubeMX

3.2.1 Налаштування портів вводу-виводу

Налаштування портів вводу виводу виконано з урахуванням рекомендацій, які було описано в попередніх розділах. Порти, які підключено до кнопок (PA15, PC6, PC8, PC9, PC11) налаштовано на зовнішнє (NVIC) переривання за переднім фронтом. Виконано підтяжку контактів кнопок до землі, щоб забезпечити високий логічний рівень на виводі МК під час

натискання кнопки. Порти, які підключено до датчика DHT11 та до дисплею встановлено у режим відкритого стоку «без підтяжки», швидкість налаштована як LOW для реалізації даного проекту цього буде цілком достатньо. Конфігурацію портів вводу-виводу наведено на рисунку 3.20.

Pin ...	Signal ...	GPIO out...	GPIO mode	GPIO Pull-up/Pull-down	Maximum output speed
PA15	n/a	n/a	External Interrupt Mode with Rising edge trigger detection	Pull-down	n/a
PC6	n/a	n/a	External Interrupt Mode with Rising edge trigger detection	Pull-down	n/a
PC8	n/a	n/a	External Interrupt Mode with Rising edge trigger detection	Pull-down	n/a
PC9	n/a	n/a	External Interrupt Mode with Rising edge trigger detection	Pull-down	n/a
PC11	n/a	n/a	External Interrupt Mode with Rising edge trigger detection	Pull-down	n/a
PD11	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE7	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE10	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE11	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE12	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE13	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE14	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low
PE15	n/a	Low	Output Open Drain	No pull-up and no pull-down	Low

Рисунок 3.20 – Налаштування портів вводу-виводу

3.2.2 Налаштування таймера TIM1

Головною метою використання цього таймера є створення функції затримки у мікросекундах, для цього достатньо вказати подільнику число 167, тоді частота таймера буде рівна 1МГц, тобто таймер буде рахувати у мікросекундах (рисунок 3.21).

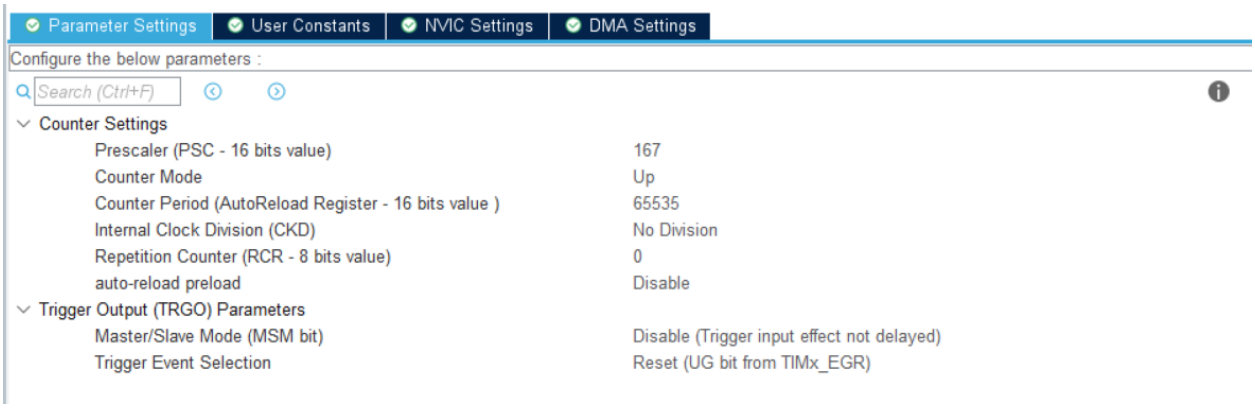

Рисунок 3.21 – Конфігурація TIM1

3.2.3 Налаштування зовнішніх переривань

Усього використовується 5 портів зовнішніх переривань – PA15, PC6, PC8, PC9, PC11. Усі порти налаштовуємо як зовнішні переривання за переднім фронтом (рисунок 3.22). Також підключаємо необхідні лінії переривань EXTI line [15:10] та EXTI line [9:5].

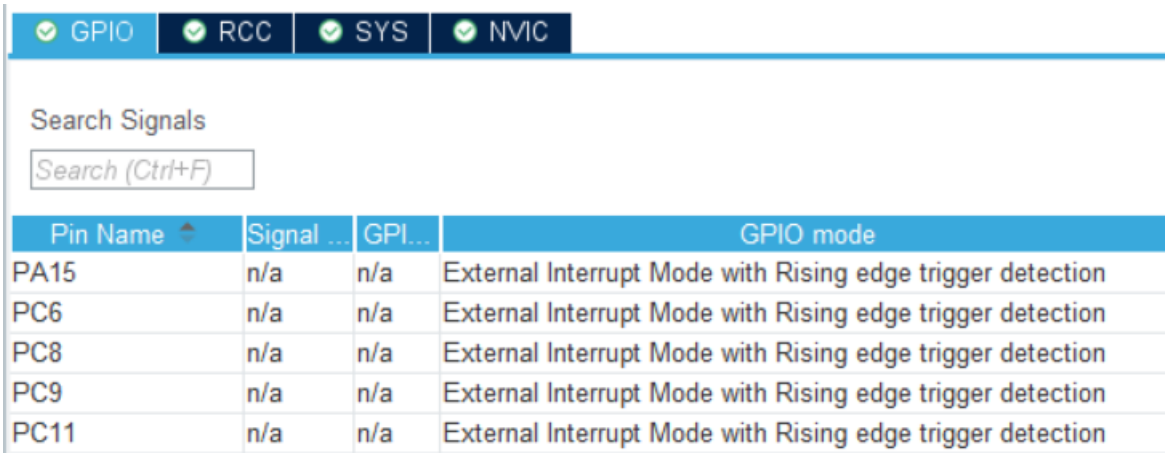

Рисунок 3.22 – Налаштування зовнішніх переривань

3.2.4 Налаштування RTC

Вмикаємо тактування та активуємо календар, у налаштуваннях встановлюємо значення Asynchronous Predivider value рівне 127 та Synchronous Predivider value рівне 249 (рисунок 3.23). Дані значення встановлюють з урахуванням тактової частоти RTC, щоб отримати частоту 1 Гц, для даного проекту RTC тактується від внутрішнього LSI джерела з частотою 32 кГц.

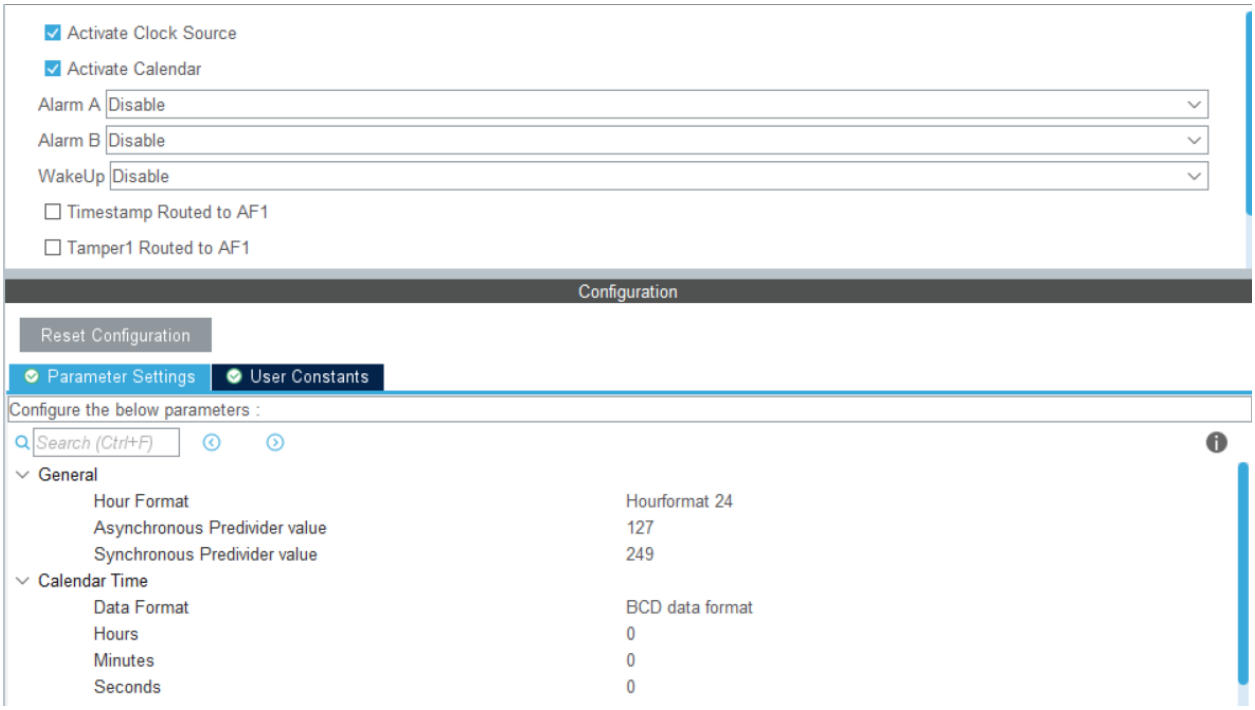

Рисунок 3.23 – Налаштування RTC

3.2.5 Налаштування тактування

У проекті МК тактується від зовнішнього кварцового резонатора HSE з частотою 8 мГц, RTC тактується від внутрішнього джерела LSI з частотою 32 кГц. Налаштування тактування наведено на рисунку 3.24.

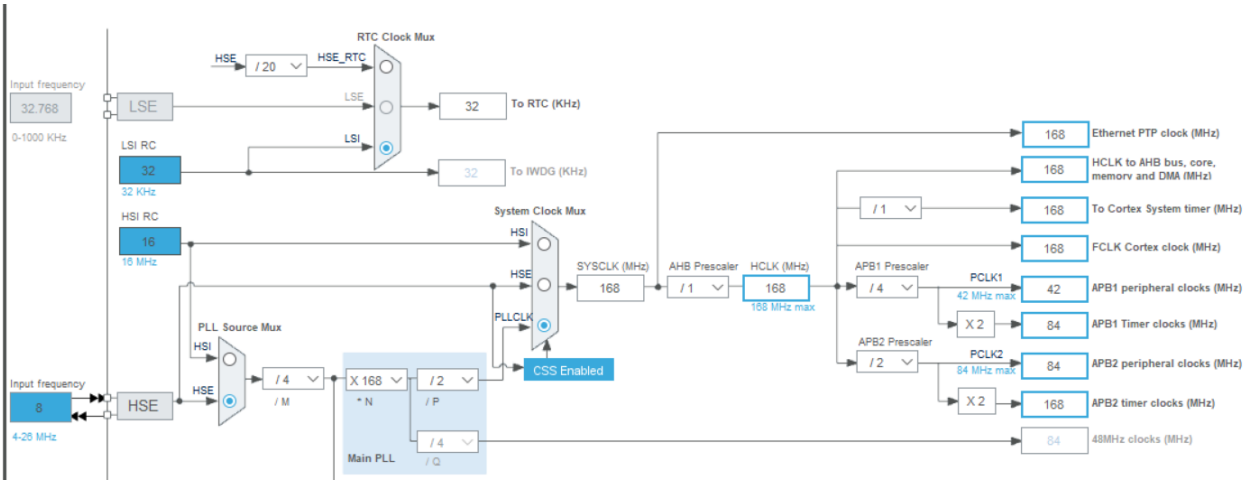

Рисунок 3.24 – Налаштування тактування

3.3 Опис алгоритму роботи

У головній програмі виконуються усі основні дії, які виконує МК, умовно програма розділена на два блоки:

- блок, який виконується лише раз;
- нескінченний цикл, виконання якого здійснюється постійно

Перший блок передбачає виконання функцій ініціалізації портів та створення глобальних змінних. Другий блок передбачає виконання вимірів та основну логіку програми.

Програму було реалізовано за допомогою мови програмування С.

Перед початком програми виконується створення глобальних змінних:

1. Змінні, які використовуються для вказання позиції символу на дисплеї.

```
int row, col = 0;
```

2. Масиви символів, які використовуються як буфер для інформації, яку необхідно вивести на дисплей.

```
char Temp[15];
char Time[15];
char Date[15];
```

3. Змінні, які зберігають значення вологості, температури та контрольної суми, вони необхідні для роботи з датчиком.

```
uint8_t RHI, RHD, TCI, TCD, SUM;
float tCelsius = 0;
float tFahrenheit = 0;
float RH = 0;
```

Початком програми є головна функція main(), яка містить у собі функції ініціалізації та нескінченний цикл. До функцій ініціалізації відносяться:

- HAL_Init() – ініціалізація бібліотеки HAL;
- SystemClock_Config() – функція налаштування режиму тактування, яка виконується згідно параметрів, що були вказані у CubeMX;
- MX_GPIO_Init() – функція виконує налаштування переривань та заповнення структури блоків GPIO, які використовуються в проекті;
- MX_RTC_Init() – функція виконує конфігурацію та налаштування RTC згідно вказаних у CubeMX параметрів;
- MX_TIM1_Init() – функція налаштування таймера;
- HAL_TIM_Base_Start(&htim1) – функція запускає відлік таймера;
- Lcd_create() – функція, яка ініціалізує об’єкт дисплею.

У нескінченному циклі виконується два основних блоки:

1. Перший блок описує логіку роботи з датчиком DHT11, спочатку встановлюється умова if(DHT11_Start()), вона ініціалізує датчик, витримуючи затримки, які було описано в попередніх розділах та очікує відповіді від нього. Якщо від датчика була отримана відповідь і він почав передачу даних, функція DHT11_Start() інтерпретується як truth і починає виконуватись частина, яка знаходиться під if умовою. Дана частина записує дані, які передає датчик та перевіряє контрольну суму, щоб впевнитись у дійсності даних і відсутності помилок. Якщо розрахована контрольна сума є правильною виміри датчика виводяться на LCD дисплей, інакше на дисплеї буде відображатися напис «Loading DHT11», що свідчить про очікування коректної роботи датчика.

2. Другий блок працює зі значеннями дати та часу. Для отримання дати та часу використано наступні функції:
   HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
   HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
   Обов'язково дотримуватись саме такої послідовності виклику функцій. Все тому, що виклик HAL_RTC_GetDate() розблокує значення в тіньових регістрах календаря вищого порядку, щоб забезпечити узгодженість між часом і датою. Читання поточного часу RTC блокує значення у тіньових регістрах календаря до зчитування поточної дати. Після отримання дати та часу, за допомогою функції sprint() виконується збереження даних у буферні змінні, які було створено на початку та вивід їх на екран дисплея.

Для реалізації затримки у мікросекундах була написана функція microDelay, у функцію передається число, що визначає кількість секунд затримки, до цього значення буде рахувати таймер. Спочатку значення таймера встановлюється в 0 і в циклі виконується реалізація затримки.

```
void microDelay (uint16_t delay)
{
 __HAL_TIM_SET_COUNTER(&htim1, 0);
 while (__HAL_TIM_GET_COUNTER(&htim1) < delay);
}
```

Функція HAL_GPIO_EXTI_Callback реалізує налаштування дати та часу за допомогою кнопок, тобто являє сообою обробник переривань, коли трапляється зовнішнє переривання, викликається ця функція. У функції виконується перевірка з якої кнопки прийшло переривання і відповідно до умови виконується інкрементування значень полів дати або часу, якщо значення після інкрементування є коректним - виконується запис у змінну та завершується виконання переривання, якщо значення є не коректним – виконується скидання значення до початкового і завершується переривання.

Алгоритм роботи програми відображено на рисунку 3.25.

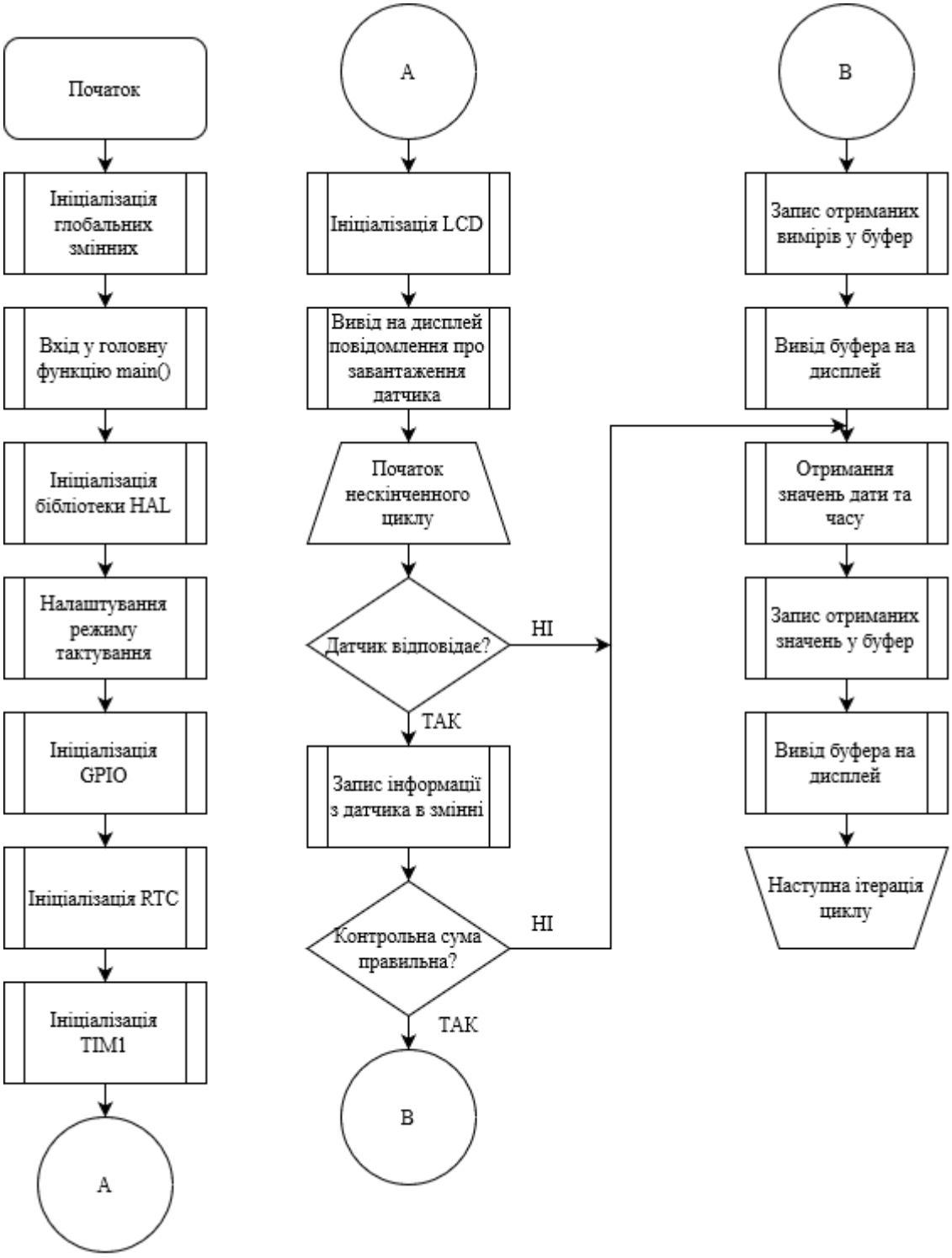

Рисунок 3.25 – Алгоритм роботи програми

РОЗДІЛ 4. ІНСТРУКЦІЯ КОРИСТУВАЧА

Для тестування створеного пристрою було використано відлагоджувальну плату STM32F407VGT Discovery та плату розширення Global Logic.

Перед подачею живлення необхідно переконатися у правильності підключення всіх складових пристрою згідно схеми.

Наступним кроком необхідно під’єднати плату до комп’ютера та завантажити програму в мікроконтролер.

Після завантаження програми рекомендується натиснути кнопку скидання на відлагоджувальній платі STM32F407VGT Discovery та пересвідчитися, що на дисплеї з’явилися початкові значення дати та часу і значення виміри температури та вологості.

Для того, аби налаштувати значення дати та часу необхідно скористатись блоком кнопок на платі розширення Global Logic. Натискання кнопок SWT4 та SWT5 призведе до зміни значень годин та хвилин відповідно. Значення будуть інкрементуватися на 1 та після досягнення граничного значення скидатись в 0. Користуючись цими двома кнопками можна налаштувати поточний час.

Для налаштування дати необхідно скористатись кнопками SWT3, SWT2, SWT1, які будуть змінювати день, місяць та рік відповідно. Значення також будуть інкрементуватися у рамках дозволеного діапазону для кожного поля дати та скидатися на початкове значення у разі досягнення граничного значення, тобто значення дня не може бути більшим за 31, значення місяця не може бути більшим за 12 та значення року не може бути більшим 99. На дисплеї відображено лише два останні символи року тому, що передбачено сприйняття за замовченням перших двох символів рівних 20, тобто 20xx.

ВИСНОВОК

Виконуючи курсовий проект було проведено повний цикл розробки пристрою, який являє собою цифровий годинник з термометром на базі мікроконтролера сімейства STM32.

У якості контролера було обрано STM32F407VGT6, у якості датчика температури – DHT11, також було використано рідкокристалічний дисплей WH1602B та кнопки для керування пристроєм.

Було створено ряд конструкторсько-технологічної документації:

- Технічне завдання
- Схема електрична принципова
- Перелік елементів (ДК91.403272.001)
- Програмне забезпечення
- Інструкцію з використання

Розроблений пристрій вийшов з достатнім функціоналом, він забезпечує користувача всіма необхідними показниками для планування та виконання різного роду роботи в комфортних умовах, адже інформування користувача про поточний час, температуру та вологість приміщення є надзвичайно важливим для продуктивної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DHT11 datasheet [Електронний ресурс] – Режим доступу до ресурсу: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. – Дата звернення: 15.11.2022
2. WH1602B datasheet [Електронний ресурс] – Режим доступу до ресурсу: https://datasheetspdf.com/pdf-file/1110065/Winstar/WH1602B/1. – Дата звернення: 12.11.2022
3. STM32F405xx STM32F407xx datasheet [Електронний ресурс] – Режим доступу до ресурсу: https://cutt.ly/U2BCc1r – Дата звернення 10.10.2022
4. Документація на плату розширення Global Logic. – С. 5.
5. How DHT11 works? Temperature and Humidity Sensor [Електронний ресурс] – Режим доступу до ресурсу: https://makersblog.in/how-dht11-works-temperature-and-humidity-sensor/. – Дата звернення 10.12.2022
6. HD44780U [Електронний ресурс] – Режим доступу до ресурсу: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf. – Дата звернення: 12.11.2022
7. STM32 Schematic [Електронний ресурс] – Режим доступу до ресурсу: https://cutt.ly/c2BBPp1 - Дата звернення: 11.11.2022
8. STM32F407xx Reference manual [Електронний ресурс] – Режим доступу до ресурсу: https://cutt.ly/D2BNaKw. – Дата звернення: 9.10.2022

ДОДАТОК А

Технічне завдання на проектування

1. **Найменування та галузь використання**

Цифровий годинник з термометром на базі мікроконтролера STM32F407VGT. Пристрій використовується у побуті для контролю за часом та значеннями температури і вологості.

2. **Підстава для розробки**

Підставою для розробки є завдання до курсового проекту, що видане викладачем відповідно до учбового плану на 7-й семестр.

3. **Мета і призначення розробки**

Метою даної роботи є розробка працюючого пристрою, набуття навичок проектування цифрової апаратури.

4. **Джерело розробки**

Пристрій розробляється вперше.

5. **Технічні вимоги**

5.1 Пристрій повинен забезпечувати:

- Моніторинг поточного значення часу та дати з подальшим виводом на дисплей;
- Можливість вимірювання температури та вологості з подальшим виводом на дисплей;
- Можливість налаштування часу та дати;
- Зручне керування

5.2 Вимоги до надійності

Середній час напрацювання на відмову повинен бути не менше 9000.

5.3 Вимоги до технологічності

Пристрій повинен бути зроблений із застосуванням широко росповсюджених технологічних процесів, та дотриманням 3-го класу точності.

5.4 Вимоги до уніфікації та стандартизації

Для виготовлення пристрою передбачається максимальне застосування стандартних і уніфікованих делатей та виробів.

5.5 Вимоги до безпеки та обслуговування

Безпека обслуговування повинна задовольняти вимогам безпеки до апаратури низької напруги згідно ГОСТ 12.2.00-75.

5.6 Умови експлуатації

Кліматичне виконання й категорія експлуатації УХЛ 4.1 за ГОСТ 15150-69.

5.7 Вимоги до елементної бази

Пристрій повинен бути спроектований з використанням, в якості обчислювального блоку, мікроконтролера STM32F407VGT. Вибір елементної бази проводити відповідно до структурної схеми з урахуванням технічих характеристик тавартості.

5.8 Вимоги до транспортування та використання

Група умов зберігання Л1 за ГОСТ 15150-69. Зберігати в зачинених, опалювальних та вентильованих приміщеннях, в яких забезпечуються наступні умови: температура повітря +5…+40ºС, відносна вологість повітря 60% при 20ºС (середньорічне значення), атмосферний тиск 84…106 кПа.

Транспортування автомобільним, залізничним або авіаційним видами траспорту в спеціальній транспортній тарі.

6. Конструкторська документація

- Пояснювальна записка
- Схема електрична принципова
- Перелік елементів
- Склад документації

7. Результати роботи

7.1 Результати даної роботи можуть бути використані як вихідна документація по створенню прототипу пристрою, його програмування й налагодження.

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

8. Робота повинна містити в собі документи

- Пояснювальну записку (формату А4, до 70 аркушів) ;
- Схему електрично принципову та перелік елементів(формату А3, А4 відповідно);
- Програмне забезпечення;
- Інструкцію по використанню;
- Додатки (формату А1-А4).

9. 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.

10. Економічні показники

В умвах даного проекту не розглядаються.

11. Етапи розробки

№ п/п	Назва етапу курсового проекту (роботи)	Час виконання етапів проекту (роботи)	Примітка
1	*Створення схеми електричної принципової*	*14.10 – 2.11*	
2	*Опис структури пристрою і його окремих складових*	*4.11 – 12.11*	
3	*Обґрунтування 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*	
8	*Подача КП до захисту*	*20.12*	

ДОДАТОК Б

Лістинг коду програми

main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "lcd.h"
#include "dht11.h"
#include "stdio.h"
#include "string.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
  RTC_TimeTypeDef sTime;
  RTC_DateTypeDef sDate;
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
```

```
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
RTC_HandleTypeDef hrtc;

TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int row, col = 0;

char Temp[15];
char Time[15];
char Date[15];

uint8_t RHI, RHD, TCI, TCD, SUM;
float tCelsius = 0;
float tFahrenheit = 0;
float RH = 0;

/* USER CODE END 0 */

/**
```

```
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */

       HAL_TIM_Base_Start(&htim1);

       //  Lcd_PortType  ports[]  =  {  D4_GPIO_Port,  D5_GPIO_Port,  D6_GPIO_Port,
D7_GPIO_Port };
  Lcd_PortType ports[] = { GPIOE, GPIOE, GPIOE, GPIOE };
  // Lcd_PinType pins[] = {D4_Pin, D5_Pin, D6_Pin, D7_Pin};
  Lcd_PinType pins[] = {GPIO_PIN_12, GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15};
  Lcd_HandleTypeDef lcd;
  //   Lcd_create(ports,   pins,   RS_GPIO_Port,   RS_Pin,   EN_GPIO_Port,   EN_Pin,
LCD_4_BIT_MODE);
```

```
  lcd = Lcd_create(ports, pins, GPIOE, GPIO_PIN_7, GPIOE, GPIO_PIN_11, LCD_4_BIT_MODE);
       Lcd_cursor(&lcd, 1,0);
       Lcd_string(&lcd, "Loading DHT11...");

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
              if(DHT11_Start())
              {
                     RHI = DHT11_Read(); // Relative humidity integral
                     RHD = DHT11_Read(); // Relative humidity decimal
                     TCI = DHT11_Read(); // Celsius integral
                     TCD = DHT11_Read(); // Celsius decimal
                     SUM = DHT11_Read(); // Check sum
                     if (RHI + RHD + TCI + TCD == SUM)
                     {
                            // Can use RHI and TCI for any purposes if whole number only
needed
                            tCelsius = (float)TCI + (float)(TCD/10.0);
                            tFahrenheit = tCelsius * 9/5 + 32;
                            RH = (float)RHI + (float)(RHD/10.0);
                            // Can use tCelsius, tFahrenheit and RH for any purposes
                            sprintf(Temp, "Temp%.1fC Hum%.0f%%", tCelsius, RH);
                            Lcd_cursor(&lcd, 1,0);
                            Lcd_string(&lcd, Temp);
                     }

              }

              HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
              HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);

              sprintf(Time, "%02d:%02d", sTime.Hours, sTime.Minutes);
              Lcd_cursor(&lcd, 0, 0);
              Lcd_string(&lcd, Time);
              sprintf(Date, "%02d.%02d.%02d", sDate.Date, sDate.Month, sDate.Year);
              Lcd_cursor(&lcd, 0, 8);
              Lcd_string(&lcd, Date);
```

```
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
```

```
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }

  /** Enables the Clock Security System
  */
  HAL_RCC_EnableCSS();
}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef sDate = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
```

```
  hrtc.Init.SynchPrediv = 249;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN Check_RTC_BKUP */

  /* USER CODE END Check_RTC_BKUP */

  /** Initialize RTC and set the Time and Date
  */
  sTime.Hours = 0x0;
  sTime.Minutes = 0x0;
  sTime.Seconds = 0x0;
  sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
  sTime.StoreOperation = RTC_STOREOPERATION_RESET;
  if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
  {
    Error_Handler();
  }
  sDate.WeekDay = RTC_WEEKDAY_MONDAY;
  sDate.Month = RTC_MONTH_JANUARY;
  sDate.Date = 0x1;
  sDate.Year = 0x0;

  if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
```

```
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 167;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
```

```
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_7|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                          |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_RESET);

  /*Configure GPIO pins : PE7 PE10 PE11 PE12
                           PE13 PE14 PE15 */
  GPIO_InitStruct.Pin = GPIO_PIN_7|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                          |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pin : PD11 */
  GPIO_InitStruct.Pin = GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : PC6 PC8 PC9 PC11 */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */
void microDelay (uint16_t delay)
{
  __HAL_TIM_SET_COUNTER(&htim1, 0);
  while (__HAL_TIM_GET_COUNTER(&htim1) < delay);
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

       if(GPIO_Pin == GPIO_PIN_9) //SWT3 button on GL board
       {
              sDate.Date++;
    if(sDate.Date >= 32) sDate.Date = 0;
       }
       else if(GPIO_Pin == GPIO_PIN_15) //SWT2 button on GL board
       {
              sDate.Month++;
    if(sDate.Month >= 13) sDate.Month = 1;
```

```
      }
      else if(GPIO_Pin == GPIO_PIN_11) //SWT1 button on GL board
      {
            sDate.Year++;
    if(sDate.Year >= 100) sDate.Year = 0;
      }
      else if(GPIO_Pin == GPIO_PIN_6) //SWT4 button on GL board
      {
            sTime.Hours++;
    if(sTime.Hours >= 24) sTime.Hours = 0;
      }
      else if(GPIO_Pin == GPIO_PIN_8) //SWT5 button on GL board
      {
            sTime.Minutes++;
            sTime.Seconds = 0;
    if(sTime.Minutes >= 60)
            {
                  sTime.Minutes = 0;
                  sTime.Seconds = 0;
            }
      }

      HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
      HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BIN);

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
```

```
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */
```

```
/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */
void microDelay (uint16_t delay);
/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

lcd.c

```
#include "lcd.h"
const uint8_t ROW_16[] = {0x00, 0x40, 0x10, 0x50};
const uint8_t ROW_20[] = {0x00, 0x40, 0x14, 0x54};
/*************************************             Static              declarations
************************************/

static void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data);
static void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command);
static void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len);


/*************************************             Function             definitions
************************************/

/**
 * Create new Lcd_HandleTypeDef and initialize the Lcd
 */
Lcd_HandleTypeDef Lcd_create(
             Lcd_PortType port[], Lcd_PinType pin[],
             Lcd_PortType rs_port, Lcd_PinType rs_pin,
             Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode)
{
       Lcd_HandleTypeDef lcd;

       lcd.mode = mode;

       lcd.en_pin = en_pin;
       lcd.en_port = en_port;

       lcd.rs_pin = rs_pin;
       lcd.rs_port = rs_port;

       lcd.data_pin = pin;
       lcd.data_port = port;

       Lcd_init(&lcd);

       return lcd;
}
```

```
/**
 * Initialize 16x2-lcd without cursor
 */
void Lcd_init(Lcd_HandleTypeDef * lcd)
{
	if(lcd->mode == LCD_4_BIT_MODE)
	{
			lcd_write_command(lcd, 0x33);
			lcd_write_command(lcd, 0x32);
			lcd_write_command(lcd, FUNCTION_SET | OPT_N);
	// 4-bit mode
	}
	else
		lcd_write_command(lcd, FUNCTION_SET | OPT_DL | OPT_N);


	lcd_write_command(lcd, CLEAR_DISPLAY);                                  //
Clear screen
	lcd_write_command(lcd, DISPLAY_ON_OFF_CONTROL | OPT_D);          //    Lcd-on,
cursor-off, no-blink
	lcd_write_command(lcd, ENTRY_MODE_SET | OPT_INC);                //  Increment
cursor
}

/**
 * Write a number on the current position
 */
void Lcd_int(Lcd_HandleTypeDef * lcd, int number)
{
	char buffer[11];
	sprintf(buffer, "%d", number);

	Lcd_string(lcd, buffer);
}

/**
 * Write a string on the current position
 */
void Lcd_string(Lcd_HandleTypeDef * lcd, char * string)
{
	for(uint8_t i = 0; i < strlen(string); i++)
```

```
	{
		lcd_write_data(lcd, string[i]);
	}
}

/**
 * Set the cursor position
 */
void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col)
{
	#ifdef LCD20xN
	lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_20[row] + col);
	#endif

	#ifdef LCD16xN
	lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_16[row] + col);
	#endif
}

/**
 * Clear the screen
 */
void Lcd_clear(Lcd_HandleTypeDef * lcd) {
	lcd_write_command(lcd, CLEAR_DISPLAY);
}

void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]){
	lcd_write_command(lcd, SETCGRAM_ADDR + (code << 3));
	for(uint8_t i=0;i<8;++i){
		lcd_write_data(lcd, bitmap[i]);
	}

}


/************************************	Static	function	definition
************************************/

/**
 * Write a byte to the command register
 */
```

```
void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command)
{
	HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_COMMAND_REG);		//
Write to command register

	if(lcd->mode == LCD_4_BIT_MODE)
	{
		lcd_write(lcd, (command >> 4), LCD_NIB);
		lcd_write(lcd, command & 0x0F, LCD_NIB);
	}
	else
	{
		lcd_write(lcd, command, LCD_BYTE);
	}

}

/**
 * Write a byte to the data register
 */
void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data)
{
	HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_DATA_REG);
	// Write to data register

	if(lcd->mode == LCD_4_BIT_MODE)
	{
		lcd_write(lcd, data >> 4, LCD_NIB);
		lcd_write(lcd, data & 0x0F, LCD_NIB);
	}
	else
	{
		lcd_write(lcd, data, LCD_BYTE);
	}

}

/**
 * Set len bits on the bus and toggle the enable line
 */
void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len)
```

```
{
	for(uint8_t i = 0; i < len; i++)
	{
		HAL_GPIO_WritePin(lcd->data_port[i], lcd->data_pin[i], (data >> i) &
0x01);
	}

	HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, GPIO_PIN_SET);
	DELAY(1);
	HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, GPIO_PIN_RESET);		//
Data receive on falling edge
}
```

lcd.h

```
#ifndef LCD_H_
#define LCD_H_

#include "stm32f4xx_hal.h"
#include "string.h"
#include "stdio.h"
#include "main.h"

// #define LCD20xN        // For 20xN LCDs
#define LCD16xN                  // For 16xN LCDs

// For row start addresses
extern const uint8_t ROW_16[];
extern const uint8_t ROW_20[];

/************************************              Command                register
************************************/
#define CLEAR_DISPLAY 0x01

#define RETURN_HOME 0x02

#define ENTRY_MODE_SET 0x04
#define OPT_S0x01                              // Shift entire display to right
#define OPT_INC 0x02                                  // Cursor increment

#define DISPLAY_ON_OFF_CONTROL 0x08
#define OPT_D0x04                              // Turn on display
#define OPT_C0x02                              // Turn on cursor
#define OPT_B      0x01                               // Turn on cursor blink

#define CURSOR_DISPLAY_SHIFT 0x10              // Move and shift cursor
#define OPT_SC 0x08
#define OPT_RL 0x04

#define FUNCTION_SET 0x20
#define OPT_DL 0x10                                   // Set interface data length
#define OPT_N 0x08                                    // Set number of display lines
#define OPT_F 0x04                                    // Set alternate font
#define SETCGRAM_ADDR 0x040
#define SET_DDRAM_ADDR 0x80                           // Set DDRAM address
```

```
/************************************ Helper macros
************************************/
#define DELAY(X) HAL_Delay(X)


/************************************ LCD defines
************************************/
#define LCD_NIB 4
#define LCD_BYTE 8
#define LCD_DATA_REG 1
#define LCD_COMMAND_REG 0


/************************************ LCD typedefs
************************************/
#define Lcd_PortType GPIO_TypeDef*
#define Lcd_PinType uint16_t

typedef enum {
	LCD_4_BIT_MODE,
	LCD_8_BIT_MODE
} Lcd_ModeTypeDef;


typedef struct {
	Lcd_PortType * data_port;
	Lcd_PinType * data_pin;

	Lcd_PortType rs_port;
	Lcd_PinType rs_pin;

	Lcd_PortType en_port;
	Lcd_PinType en_pin;

	Lcd_ModeTypeDef mode;

} Lcd_HandleTypeDef;
```

```
/************************************ Public functions
************************************/
void Lcd_init(Lcd_HandleTypeDef * lcd);
void Lcd_int(Lcd_HandleTypeDef * lcd, int number);
void Lcd_string(Lcd_HandleTypeDef * lcd, char * string);
void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col);
Lcd_HandleTypeDef Lcd_create(
            Lcd_PortType port[], Lcd_PinType pin[],
            Lcd_PortType rs_port, Lcd_PinType rs_pin,
            Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode);
void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]);
void Lcd_clear(Lcd_HandleTypeDef * lcd);

#endif /* LCD_H_ */
```

dht11.c

```
#include "dht11.h"

uint32_t pMillis, cMillis;

uint8_t DHT11_Start (void)
{
  uint8_t Response = 0;
  GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
  GPIO_InitStructPrivate.Pin = DHT11_PIN;
  GPIO_InitStructPrivate.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as output
  HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, GPIO_PIN_RESET);   // pull the pin low
  HAL_Delay(20);   // wait for 20ms
  HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, GPIO_PIN_SET);   // pull the pin high
  microDelay (30);   // wait for 30us
  GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
  GPIO_InitStructPrivate.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as input
  microDelay (40);
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
  {
    microDelay (80);
    if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
```

```
  }
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {
    cMillis = HAL_GetTick();
  }
  return Response;
}

uint8_t DHT11_Read (void)
{
  uint8_t a,b;
  for (a=0;a<8;a++)
  {
    pMillis = HAL_GetTick();
    cMillis = HAL_GetTick();
    while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
    {  // wait for the pin to go high
      cMillis = HAL_GetTick();
    }
    microDelay (40);   // wait for 40 us
    if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
      b&= ~(1<<(7-a));
    else
      b|= (1<<(7-a));
    pMillis = HAL_GetTick();
    cMillis = HAL_GetTick();
    while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
    {  // wait for the pin to go low
      cMillis = HAL_GetTick();
    }
  }
  return b;
}
```

dht11.h

```
#ifndef DHT11_H_
#define DHT11_H_

#include "main.h"

#define DHT11_PORT GPIOD
#define DHT11_PIN GPIO_PIN_11

uint8_t DHT11_Start (void);
uint8_t DHT11_Read (void);

#endif /* LCD_H_ */
```