Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: **Методи та програмно-апаратні засоби керування сонячним колектором для регулювання температурою води в басейні**

Виконав(ла): студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

_____ *Невмержицький В. В.*
(підпис) (прізвище та ініціали)

Керівник _____ *Стадник Н. Б.*
(підпис) (прізвище та ініціали)

Нормоконтроль _____ *Тиш Є. В.*
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ *Осухівська Г. М.*
(підпис) (прізвище та ініціали)

Рецензент _____ Бойко І. В.
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г. М.*
(підпис) (прізвище та ініціали)
«17» листопада 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Невмержицькому Віталію Володимировичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи та програмно-апаратні засоби керування сонячним колектором для регулювання температурою води в басейні*

Керівник роботи *Стадник Наталія Богданівна, к.т.н., старший викладач*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *листопада* 2025 року № 4/7-987

2. Термін подання студентом завершеної роботи *16 грудня 2025 р.*

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ
1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Охорона праці та безпека в надзвичайних ситуаціях
Висновки

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

1. Актуальність і мета дослідження.
2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3. Структурна схема
4. Схема електрична принципова блока 1
5. Схема електрична принципова блока 2
6. Блок-схема блока 1
7. Блок-схема блока 2
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., зав.каф. КС*	*04.12.2025*	
Безпека в надзвичайних ситуаціях	*Стручок В. С. ст. викл. каф.*	22.12.2025	

7. Дата видачі завдання *17.11.2025р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Огляд літературних джерел. Постановка завдання на кваліфікаційну роботу.*	*17.11.2025р. – 23.11.2025р.*	*Викон.*
2.	*Робота над другим розділами: Розробка апаратної частини системи контролю*	*24.11.2025р. – 02.12.2025р.*	*Викон.*
3.	*Робота над третім розділами: Розробка програмного забезпечення системи контролю*	*03.12.2025р. – 10.12.2025р.*	*Викон.*
4.	*Охорона праці та безпека в надзвичайних ситуаціях*	*04.12.2025р. – 10.12.2025р.*	*Викон.*
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*11.12.2025р. – 19.12.2025р.*	*Викон.*
6.	*Попередній захист кваліфікаційної роботи магістра*	*16.12.2025р.*	*Викон.*
7.	*Захист кваліфікаційної роботи магістра*	*26.12.2025р.*	*Викон.*

Студент _____ *Невмержицький В. В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Стадник Н. Б.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Невмержицький Віталій Володимирович. Методи та засоби керування сонячним колектором для управління температурою води в басейні робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук.кер. Стадник Наталія Богданівна Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025.

Ключові слова: контроль температури, енергоефективність, сонячний колектор, автоматизована система керування, мікроконтролер, датчики температури.

Сучасні тенденції переходу до використання відновлюваних джерел енергії створюють потребу у впровадженні автоматизованих систем керування технологічними процесами, зокрема систем нагріву води на основі сонячних колекторів. Ефективність роботи таких установок значною мірою залежить від точності вимірювання температурних параметрів, коректної оцінки теплових потоків. Це особливо актуально для системи підігріву води в басейні, де необхідно забезпечувати стабільний і безпечний температурний режим за мінімальних витрат енергії.

Розроблена система контролю сонячного колектора базується на мікроконтролері STM32, який здійснює зчитування, обробку та аналіз даних від цифрових датчиків температури DS18B20, розташованих у ключових точках: на вході та виході колектора, у баку-накопичувачі та в басейні.

Алгоритм керування ґрунтується на аналізі різниць температур між входом і виходом колектора і між басейном та баком. У разі, коли температура в баці перевищує температуру у басейні на встановлену користувачем величину, мікроконтролер активує реле для вмикання насоса басейна. Коли різниця температур зменшується до нижнього порогу, насос басейна вимикається, що запобігає небажаному охолодженню бака та підвищує енергоефективність.

ANNOTATION

Nevmerzhytskyi Vitaliy Volodymyrovych Methods and hardware–software tools for controlling a solar collector to regulate swimming-pool water temperature. Master's Graduation Thesis: speciality 123 — Computer engineering / supervisor Stadnik Natalia Bogdanivna Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Key words: temperature control, energy efficiency, solar collector, automated control system, microcontroller, temperature sensors.

Modern trends in the transition to the use of renewable energy sources create a need for the implementation of automated process control systems, in particular water heating systems based on solar collectors. The efficiency of such installations largely depends on the accuracy of measuring temperature parameters, correct assessment of heat flows. This is especially important for a water heating system in a swimming pool, where it is necessary to ensure a stable and safe temperature regime with minimal energy consumption.

The developed solar collector control system is based on the STM32 microcontroller, which reads, processes and analyzes data from digital DS18B20 temperature sensors located at key points: at the collector inlet and outlet, in the storage tank and in the pool.

The control algorithm is based on the analysis of temperature differences between the collector inlet and outlet and between the pool and the tank. When the temperature in the tank exceeds the pool temperature by a user-set value, the microcontroller activates a relay to turn on the pool pump. When the temperature difference decreases to the lower threshold, the pool pump turns off, which prevents unwanted cooling of the tank and increases energy efficiency.

ЗМІСТ

ВСТУП

Актуальність роботи. Зростання ролі відновлюваних джерел енергії та вимог до енергоефективності зумовлює широке впровадження сонячних колекторів, зокрема для підігріву води в басейнах, що дає змогу зменшити споживання традиційних енергоресурсів і експлуатаційні витрати. Ефективність таких систем значною мірою залежить від рівня автоматизації контролю температурних режимів. Недостатня автоматизація спричиняє теплові втрати та зниження надійності обладнання, тому розроблення сучасних програмно-апаратних засобів контролю температури води є актуальним науково-прикладним завданням.

Мета кваліфікаційної роботи. Метою кваліфікаційної роботи є розроблення та обґрунтування методів і програмно-апаратних засобів контролю сонячного колектора для автоматизованого регулювання температури води в басейні на основі мікроконтролерних технологій.

Завдання кваліфікаційної роботи. Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

– виконати аналіз існуючих методів та засобів контролю температури в системах сонячного нагріву води;

– розробити структурну та функціональну схеми системи контролю і керування сонячним колектором;

– обґрунтувати вибір елементної бази та програмно-апаратних компонентів системи;

– реалізувати алгоритми автоматичного регулювання температури води та оцінити ефективність роботи розробленої системи.

Об’єкт дослідження – процес автоматизованого контролю та керування температурою води в басейні з використанням сонячного колектора.

Предмет дослідження – методи, алгоритми та програмно-апаратні засоби реалізації системи контролю і регулювання температури води на базі мікроконтролерів.

Методи дослідження: У кваліфікаційній роботі використано методи системного аналізу для дослідження структури та принципів роботи системи сонячного нагріву, методи математичного та алгоритмічного моделювання для розроблення логіки керування, а також методи програмної інженерії та експериментального аналізу для реалізації і перевірки працездатності програмно-апаратного комплексу.

Наукова новизна дослідження: Наукова новизна роботи полягає в удосконаленні підходів до автоматизованого контролю сонячного колектора шляхом застосування диференційних алгоритмів регулювання температури та розподіленої мікроконтролерної архітектури, що дозволяє підвищити енергоефективність і надійність роботи системи.

Практичне значення результатів кваліфікаційної роботи: Практичне значення отриманих результатів полягає у можливості використання розробленої системи контролю для автоматичного підтримання заданої температури води в басейнах приватного та громадського призначення. Запропоновані програмно-апаратні рішення можуть бути використані під час проєктування енергоефективних систем підігріву води та в навчальному процесі з дисциплін комп’ютерної інженерії.

Публікації. Результати дослідження апробовано на XIII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» та XIV міжнародній науково-технічній конференції молодих учених та студентів «Актуальні задачі сучасних технологій» у вигляді тез доповідей.

Структура роботи. Кваліфікаційна робота складається з пояснювальної записки та графічної частини. Пояснювальна записка містить вступ, чотири розділи, висновки, список використаних джерел та додатки.

РОЗДІЛ 1
АНАЛІЗ ТЕОРЕТИЧНИХ ТА ТЕХНІЧНИХ ОСНОВ КОНТРОЛЮ СОНЯЧНИМИ КОЛЕКТОРАМИ

Сонячні колектори є ефективним рішенням для нагріву води, зокрема в басейнах, оскільки забезпечують економію традиційних енергоносіїв і зменшення викидів парникових газів.

Сонячний колектор перетворює сонячне випромінювання на теплову енергію та використовується в системах гарячого водопостачання, опалення і підігріву басейнів, а його ключовим елементом є абсорбер із селективним покриттям.

Ефективність роботи таких систем залежить не лише від конструкції колектора, а й від системи контролю, яка забезпечує оптимальні режими роботи, захист від перегріву та адаптацію до погодних умов із застосуванням сучасних алгоритмів керування.

1.1. Принципи роботи сонячного колектора і теплові баланси басену

Сонячний колектор — це пристрій, який перетворює сонячне випромінювання на теплову енергію. Основним елементом колектора є абсорбер, який завдяки спеціальному селективному покриттю ефективно поглинає сонячні промені і передає тепло робочому середовищу — воді або рідині-носію.

Принцип роботи колектора (рис. 1.1) базується на декількох послідовних процесах:

- Уловлювання сонячного світла: колектор встановлюється так, щоб на його поверхню потрапляло максимальне сонячне випромінювання, зазвичай на даху або іншій відкритій поверхні.
- Поглинання енергії: темна поверхня абсорбера, розташована під прозорим покриттям, ефективно поглинає сонячні промені та нагрівається.

- Передача тепла: нагрітий абсорбер передає тепло теплоносію через систему трубок, підвищуючи його температуру.
- Циркуляція теплоносія: гаряча рідина рухається по системі труб для передачі тепла до бойлера або теплоакумулятора.
- Зберігання тепла: теплоносій передає тепло воді у баку, яка потім використовується для гарячого водопостачання або опалення.
- Повторення циклу: охолоджена рідина повертається до колектора для повторного нагріву, і цикл триває.

Рис. 1.1. Принцип роботи сонячного колектора

Ефективність сонячного колектора визначається його здатністю перетворювати до 75% сонячного випромінювання в корисне тепло за оптимальних умов.

Циркуляція теплоносія може бути природною (термосифонною), що не потребує насоса й застосовується переважно в сезонних системах, або примусовою — з використанням насоса та антифризу, що забезпечує вищу ефективність і гнучкість розміщення елементів системи [1].

Оцінка теплової віддачі колектора базується на рівняннях теплообміну з урахуванням теплових втрат басейну та метеорологічних чинників. Робота

колектора є доцільною лише за умови перевищення його температури над температурою басейну з урахуванням втрат, інакше робота насоса неефективна.

1.2. Класифікація сонячних колекторів та особливості їх використання для підігріву басейнів

Сонячні колектори є ключовим елементом систем нагріву води та відрізняються за конструкцією й умовами експлуатації. Вибір типу колектора визначає ефективність системи, особливо в басейнах, де важливими є робота з великими об’ємами води, стабільна температура та мінімальні теплові втрати.

Найбільш поширеними у практиці сонячного водонагріву є плоскі та вакуумні сонячні колектори. Плоскі сонячні колектори мають відносно просту конструкцію, що включає абсорбційну поверхню, теплообмінні канали, теплоізоляційний шар і прозоре захисне покриття (рис. 1.2.).

Рис. 1.3. Конструкція плоского сонячного колектора

Принцип їх роботи полягає у поглинанні сонячного випромінювання абсорбером та передачі отриманого тепла теплоносію. Такі колектори

забезпечують нагрів води до температур, достатніх для систем підігріву басейнів, і характеризуються надійністю, доступною вартістю та простотою експлуатації.

Завдяки здатності ефективно працювати за порівняно невисоких температур теплоносія плоскі сонячні колектори є доцільними для використання в басейнах середнього та великого об’єму. Їх застосування дозволяє компенсувати теплові втрати, спричинені випаровуванням води та тепловіддачею в навколишнє середовище, без необхідності створення високих температурних режимів.

Вакуумні сонячні колектори відрізняються складнішою конструкцією, яка передбачає наявність вакуумного простору між стінками трубок.

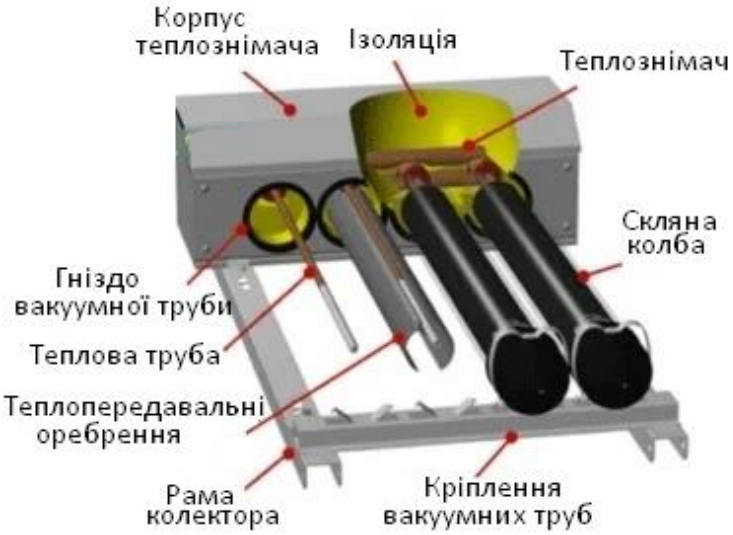

Рис. 1.4. Загальна конструкція вакуумного трубчастого сонячного колектора

Вакуумні сонячні колектори мають високий коефіцієнт корисної дії і здатні забезпечувати високі температури, однак для підігріву басейнів це часто є надлишковим і економічно недоцільним через вищу вартість та складність систем керування.

Окрім водяних сонячних колекторів, існують також повітряні та спеціалізовані типи колекторів, які застосовуються переважно у вентиляційних або технологічних системах. У системах підігріву басейнів такі колектори застосовуються обмежено через низьку ефективність. Тому для підігріву води в

басейнах найбільш доцільним є використання плоских сонячних колекторів, які поєднують ефективність, надійність і економічну доцільність та обрані як базові для подальшої розробки системи керування.

1.3. Методи контролю температури води в системах сонячного нагріву

В Україні застосовуються національні стандарти, що ідентичні міжнародним (ДСТУ EN 12975, ДСТУ ISO 9806 тощо), які регламентують основні параметри та методи випробувань колекторів і визначення їхніх теплових характеристик. Ці документи також важливі при виборі робочих температур, граничних режимів і заходів захисту від перегріву [9].

Виділяють прості (рефлексивні), середні та складні методи керування. До простих методів належать:

- Ручне керування – користувач самостійно вмикає або вимикає насос, орієнтуючись на власні спостереження (наприклад, коли сонячно — вмикає, увечері — вимикає). Переваги: найпростіша реалізація, не потребує електроніки чи датчиків. Недоліки: залежність від людського фактору, неекономічність, ризик перегріву або охолодження води.
- Часове (розкладне) керування — насос або циркуляція вмикаються за заздалегідь заданим графіком (наприклад, 9:00–17:00). Плюси: простота впровадження, не потрібні датчики температури; мінуси: неекономічність у змінних погодних умовах (вкл. у похмурі дні) і ризик охолодження в нічний час. Часто використовується для простих басейнових установок з постійною сезонною експлуатацією.
- Порогове (термостатне) керування — увімкнення/вимкнення при досягненні фіксованого значення T_pool або T_collector. Проста альтернатива диференційному контролю, але може створювати часті цикли вмикання/вимкнення (short-cycling) і не враховує різниці температур між колектором і басейном [10].

Диференційний (ΔT) контроль належить до середнього рівня складності— найбільш розповсюджений для басейнів – насос вмикається, коли різниця температур T_collector − T_pool перевищує певний поріг ΔT_on, і вимикається, коли різниця опуститься нижче ΔT_off (ΔT_off < ΔT_on — для гістерезису). Така схема гарантує, що насос перекачує тепло лише коли колектор ефективно віддає його воді. Переваги: проста логіка, висока практична ефективність для відкритих колекторів басейнів, мінімізація марного енергоспоживання насосом. Недоліки: не враховує зміни інсоляції, втрат тепла від випаровування, і може працювати не оптимально при великих запасах води [11].

До складних (інтелектуальних) методи контролю, що використовують математичні моделі, зворотний зв’язок, прогнозування або адаптацію для підтримання оптимальних умов, належать:

- Пропорційно-інтегрально-диференційний (PID) контроль – використовує PID-регулятор, що повертає неперервний сигнал (або широтно-імпульсну модуляцію для клапанів/помп), це дозволяє плавно регулювати потік і досягати заданої температури без сильних коливань.

Коли застосовувати: системи з теплообмінниками, коли потрібне точне утримання T або в установках з підвищеними вимогами до комфорту. Переваги: менше перерегулювань, більша точність; недоліки: потребує ідентифікації динамічної моделі, налаштування коефіцієнтів (P,I,D), чутливий до зміни параметрів системи.

- Нечітка логіка та адаптивні методи – полягає у застосуванні нечіткої керованості або адаптивних алгоритмів дозволяє інтерпретувати нечіткі вхідні дані (зміни інсоляції, часткова хмарність) і підлаштовувати правила керування «на льоту».

- Прогнозні та оптимізуючі методи (MPC — Model Predictive Control) – MPC використовує модель процесу і прогноз зовнішніх змін (інсоляція, температура повітря) для оптимального планування дій (керування насосом, перемиканням контурів) на горизонті часу з урахуванням обмежень (максимальна температура, економія енергії).

Переваги: можливість мінімізувати витрати або уникнути перегріву (сталінга), працює краще при мультиоб’єктних системах (декілька джерел тепла). Недоліки: потребує моделі, обчислювальних ресурсів і даних прогнозу; у дослідженнях українських вишів розглядається як перспективний підхід для мікроенергетичних систем і розподіленого керування [12].

Доречно використовувати захисні алгоритми та практичні заходи, такі як антиперегрів та антициклування:

- Граничні захисти: обмеження максимальної температури в баку/колекторі (наприклад, автоматичне відключення насоса або відкриття байпасу при T > Tmax). Рекомендується застосовувати апаратний рівень контролю як резерв (термостатичні клапани, механічні запобіжники) поряд із програмним.
- Анти-циклування: вводяться мінімальні часи включення/виключення або лічильники циклів, щоб уникнути частих вмикань насосів. Стандартні контролери часто мають таймери блокування після відключення.

Антициклування в системах сонячного нагріву реалізується шляхом введення мінімальних часів увімкнення та вимкнення або обмеження кількості циклів роботи насосів, що запобігає їх частим пускам; стандартні контролери зазвичай мають таймери блокування після відключення.

Для ефективного автоматичного регулювання температури застосовують температурні датчики, які контролюють стан теплоносія в колекторі, зворотному контурі та воді басейну.

Найчастіше використовують терморезистивні датчики (Pt100, Pt1000) і термістори (NTC), а в промислових рішеннях — термопари (типу K або J). Коректність вимірювань забезпечується правильним розміщенням датчиків у зонах активної циркуляції та обов’язковим калібруванням і захистом від впливу вологи та сонячного випромінювання [13].

Для забезпечення стабільного функціонування системи контролю температури у сонячному колекторі необхідна надійна апаратна база, яка включає комплекс вимірювальних, керувальних та виконавчих пристроїв.

Апаратна частина системи реалізує збирання інформації від датчиків, оброблення сигналів, формування керуючих дій та передавання даних до виконавчих механізмів — насосів, клапанів або контролерів.

Основними компонентами апаратної системи є:

- Мікроконтролер (керуючий модуль) Центральним елементом системи є мікроконтролер, який виконує функції збору, аналізу та оброблення даних від датчиків температури.

У сучасних рішеннях часто використовуються плати Arduino Uno, ESP32, STM32 або Raspberry Pi Pico, завдяки їхній доступності, низькому енергоспоживанню та широкій підтримці сенсорів.

Контролер виконує обчислення згідно з алгоритмом контролю (наприклад, PID-регулювання) і формує сигнали для керування насосом або електромагнітним клапаном.

- Датчики температури, використовуються терморезистори Pt100 / Pt1000 або термістори NTC 10K, які підключаються до аналогових входів контролера через схеми узгодження. У випадку цифрових датчиків, таких як DS18B20, передача даних здійснюється по шині 1-Wire, що спрощує монтаж.
- Підсилювачі та аналого-цифрові перетворювачі (АЦП).

Для забезпечення точності вимірювання сигналів від аналогових датчиків застосовуються операційні підсилювачі або вбудовані АЦП мікроконтролера. Високоточні системи можуть додатково використовувати зовнішні АЦП типу ADS1115.

- Виконавчі елементи.

Керування подачею теплоносія реалізується за допомогою циркуляційного насоса, яким керує контролер через твердотільне реле (SSR) або MOSFET-транзистор. У більш складних системах застосовують ШІМ-регулювання (PWM) для плавної зміни швидкості обертання насоса.

- Блок живлення.

Система живиться від джерела постійної напруги 12 або 24 В. Для підвищення надійності часто додають резервне живлення (акумулятор або

сонячну батарею з контролером заряду), що забезпечує автономну роботу при відключенні електромережі.

- Інтерфейси та відображення даних.

Для зручності користувача система може містити:

- LCD-дисплей (16×2 або 20×4) для виведення температурних показників;
- світлодіодні індикатори стану насоса чи аварійних режимів;
- кнопки або енкодер для зміни налаштувань.

У сучасних реалізаціях можливе підключення модуля Wi-Fi (ESP8266 / ESP32) для моніторингу параметрів через мобільний додаток або веб-інтерфейс [14][15][16].

Принцип роботи апаратної системи:

- Датчики температури постійно вимірюють температуру води у колекторі, у басейні та в зворотному контурі.
- Сигнали надходять до мікроконтролера, де відбувається аналіз різниці температур (ΔT) та порівняння з заданими пороговими значеннями.
- У разі перевищення допустимої різниці контролер вмикає насос, забезпечуючи циркуляцію теплоносія до басейну.
- Коли температура води досягає встановленої межі, контролер автоматично вимикає насос або зменшує його продуктивність.
- У разі аварійних ситуацій (перегрів, відсутність сигналу від датчика) система переходить у безпечний режим, вимикаючи виконавчі елементи.

Особливості апаратної реалізації:

- Усі з’єднання виконуються із герметичним захистом (IP65–IP68) для безпечної роботи в умовах підвищеної вологості.
- Встановлюються запобіжники та стабілізатори напруги, щоб уникнути виходу з ладу електроніки.
- Для мінімізації втрат енергії система споживає невеликий струм у режимі очікування (<50 мА).

- Для підвищення точності контролю застосовується періодична калібровка датчиків та перевірка відповідності показників [17].

1.4. Аналіз існуючих систем автоматизації та моніторингу

Сучасні системи сонячного нагріву води оснащуються автоматизованими засобами контролю та моніторингу, що підвищують енергоефективність, зменшують теплові втрати та забезпечують стабільну роботу обладнання. Автоматизація таких систем базується на безперервному зборі й обробці даних про стан колектора та керуванні насосами і клапанами відповідно до поточних умов роботи.

Системи автоматизації сонячних колекторів умовно поділяють на три рівні складності які наведені в табл. 1.1.

Таблиця 1.1

Рівні складності систем автоматизації сонячних колекторів

Рівень	Тип системи	Характеристика
Прості	Механічні або електронні термостати	Автоматично вмикають насос при досягненні заданої температури; не мають зворотного зв’язку
Середнього рівня	Мікроконтролерні системи з двома датчиками	Використовують алгоритми ΔT-контролю, обробляють інформацію з двох сенсорів (колектор і резервуар).
Високотехнологічні (інтелектуальні)	Комплексні системи з мікропроцесором, дисплеєм, Wi-Fi-модулем, хмарним моніторингом	Реалізують PID-регулювання, архівування даних, дистанційний контроль через мобільний застосунок або вебінтерфейс

На сьогоднішній день в Україні застосовуються декілька сучасних систем автоматизації й моніторингу:

- Система «Теплоконтроль SLR-03» (ТОВ «Теплосфера», м. Київ).

Український контролер, призначений для керування сонячними колекторами в системах гарячого водопостачання. Використовує два температурні датчики (колектор і бак). Алгоритм контролю побудований за принципом диференційного порівняння ΔT. Містить індикаторний дисплей, ручний і автоматичний режими, а також вихід для керування насосом через реле. Недолік: відсутність можливості віддаленого моніторингу через інтернет.

- Система «Heliomatic HM-2» (м. Львів, ПП «Екопанель Україна»)

Це автоматичний контролер для керування двома контурами сонячного нагріву — колектором та буферною ємністю. Використовує до чотирьох датчиків температури. Підтримує функції захисту від перегріву, антизамерзання, нічного охолодження. Має вбудований модуль реєстрації даних на SD-карту та можливість підключення до ПК через USB [18].

- Система на базі Arduino/ESP32 (навчально-демонстраційна, КПІ ім. І. Сікорського).

На кафедрі автоматизації енергетичних процесів розроблено відкриту систему контролю сонячного колектора, побудовану на мікроконтролері Arduino Mega із застосуванням Wi-Fi-модуля ESP8266. Датчики DS18B20 вимірюють температуру на вході, виході та в басейні. Дані передаються на веб-сервер, що дозволяє моніторинг у реальному часі через смартфон. Алгоритм контролю реалізовано на основі PID-регулятора з автоматичним підстроюванням коефіцієнтів. Використовується для навчальних цілей і тестування енергоефективності [19].

- Промислова система «SOLAR-LOG 1200» (імпорт, використовується в Україні).

Використовується у великих готельних або спортивних комплексах із сонячними установками. Здійснює збір даних з десятків датчиків, моніторинг у хмарному сервісі. Може керувати насосами, клапанами, обігрівачами, реєструвати

споживання енергії. Інтерфейс користувача реалізовано у вигляді вебпанелі з графіками та аналітикою [20].

Порівняльна характеристика існуючих систем наведена в табл. 1.2.

Таблиця 1.2

Порівняльна характеристика систем автоматизації та моніторингу

Параметр	Теплоконтроль SLR-03	Heliomatic HM-2	Arduino/ESP32	Solar-Log 1200
Тип контролю	ΔT-контроль	ΔT, PID	PID	Інтелектуальний, прогнозний
Кількість датчиків	2	4	3-5	до 30
Інтерфейс користувача	LCD	LCD, кнопки	Вебінтерфейс	Веб + мобільний
Можливість віддаленого моніторингу	Ні	Частково (через ПК)	Так	Так
Рівень складності	Середній	Середній	Високий	Високий
Вартість (станом на 2024 р.)	≈ 2000 грн	≈ 4000 грн	≈ 1500–2500 грн	> 15 000 грн

Проведений аналіз показує, що на ринку України доступні як спрощені бюджетні рішення, так і високотехнологічні інтелектуальні системи, здатні забезпечити глибокий моніторинг і керування процесом нагріву води.

Основні тенденції розвитку систем автоматизації такі:

- Перехід від локальних контролерів до віддалених моніторингових платформ.

- Використання мікроконтролерів з бездротовими модулями (ESP32, GSM, LoRa).

- Реалізація PID- та адаптивних алгоритмів для стабільності температурного режиму.

- Інтеграція з мобільними застосунками та хмарними сервісами для зручності користувача [21].

1.5. Висновки до розділу

У ході проведеного аналізу теоретичних і технічних основ функціонування сонячних колекторів було з’ясовано, що сучасні системи перетворення сонячної енергії є одним із найефективніших напрямів використання відновлюваних джерел енергії. Принцип дії сонячного колектора (рис.1.5), ґрунтується на поглинанні сонячного випромінювання абсорбером і передачі теплової енергії робочому середовищу, яке циркулює системою теплообміну.

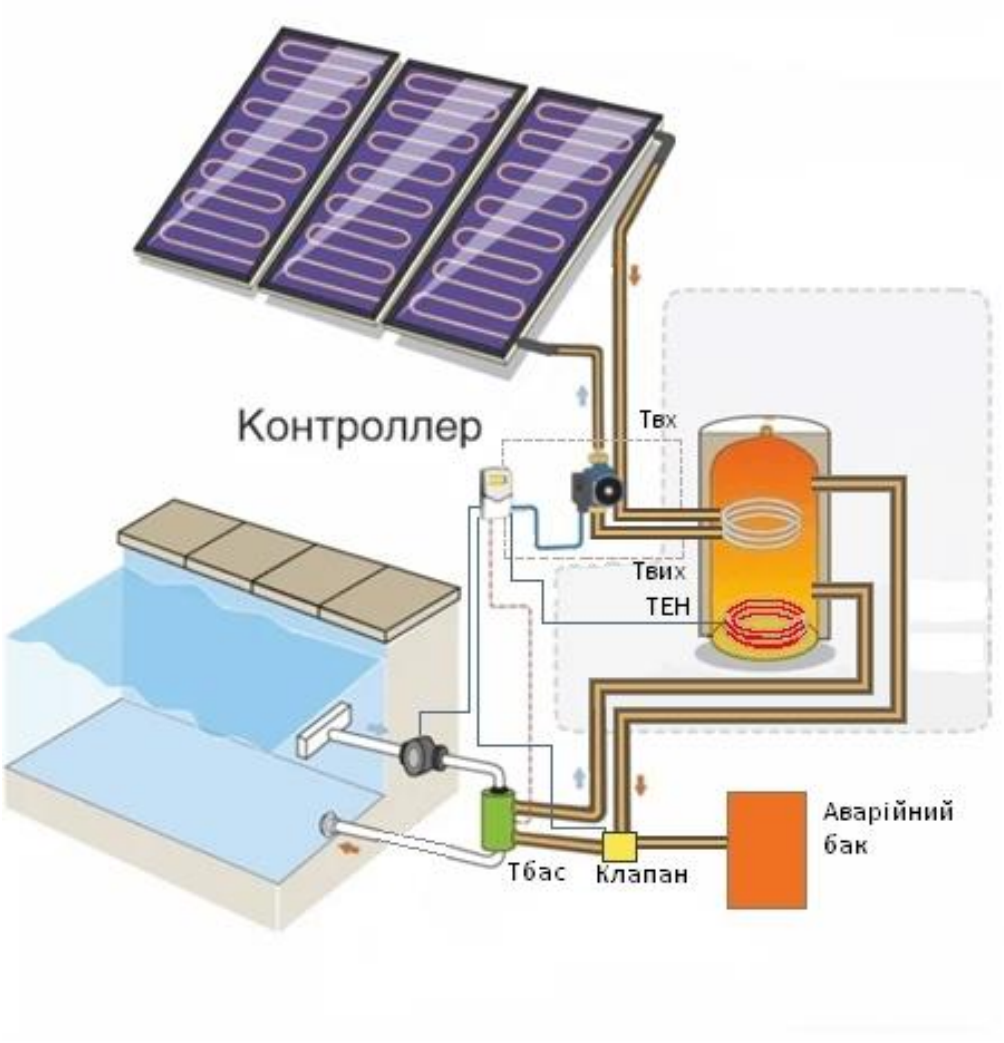

Рис. 1.5. Система контролю сонячного колектора

Розглянуто основні типи сонячних колекторів - плоскі, вакуумні трубчасті, гібридні, термосифонні, повітряні та спеціалізовані колектори для басейнів.

Встановлено, що вибір конкретного типу колектора визначається кліматичними умовами, потребами споживача, місцем установки та фінансовими можливостями. Найбільш поширеними у практичному застосуванні є плоскі та вакуумні трубчасті колектори, які забезпечують високу ефективність перетворення сонячної енергії в тепло.

Важливою складовою ефективної роботи геліосистем є система контролю та автоматизації, яка забезпечує оптимальне керування температурними режимами, знижує теплові втрати та підвищує надійність експлуатації. Без системи контролю робота колектора не може бути стабільною, особливо при змінних погодних умовах. Саме тому питання розроблення ефективних методів і засобів контролю температури води в системах сонячного нагріву є актуальним напрямом у галузі комп’ютерної інженерії та автоматизації.

Таким чином, проведений аналіз дозволив визначити, що подальше дослідження має бути спрямоване на розроблення апаратно-програмних рішень для реалізації системи контролю сонячного колектора яка забезпечить стабільний температурний режим води в басейні та підвищить загальну енергоефективність системи.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ КОНТРОЛЮ ТА УПРАВЛІННЯ СОНЯЧНИМ КОЛЕКТОРОМ

2.1. Постановка задачі управління

Зі зростанням вартості енергоресурсів ефективність систем сонячного теплопостачання значною мірою залежить від надійного контролю робочих параметрів і захисту від аварійних режимів. Тому актуальним є створення автоматизованої системи керування сонячним колектором, здатної підтримувати задану температуру води в басейні та забезпечувати безпечну роботу геліосистеми.

Метою розробки є створення мікропроцесорної системи контролю, яка на основі постійного моніторингу температур у ключових точках та витрати теплоносія забезпечує оптимальну теплопередачу, енергоефективну роботу та реалізацію механізмів захисту від перегріву.

Система керування побудована за розподіленою архітектурою та включає два мікроконтролери. Головний контролер STM32F411CEU6 виконує збір і аналіз даних, взаємодію з користувачем, збереження інформації у зовнішній пам’яті та обмін даними з віддаленим модулем через радіоканал NRF24L01. Віддалений контролер STM32F103C8T6 здійснює зчитування даних із датчиків температури DS18B20 і витратоміра YF-S201, а також керує виконавчими механізмами, зокрема насосами, електричним нагрівачем і аварійним клапаном.

Основними завданнями системи є вимірювання температур у декількох зонах, контроль витрати теплоносія, передавання та обробка даних між контролерами, реалізація алгоритмів керування на основі температурних різниць ΔT, а також забезпечення захисту від перегріву й аварійних режимів. Додатково реалізовано візуалізацію параметрів роботи системи, реєстрацію історії вимірювань та механізми підвищення енергоефективності.

У результаті розробляється автономна система керування сонячним колектором, яка забезпечує автоматичне підтримання температури води в басейні, підвищує надійність і довговічність обладнання та зменшує потребу в ручному втручанні користувача.

2.2. Вимоги до системи контролю

Система контролю сонячного колектора повинна відповідати комплексу функціональних, технічних та економічних вимог, які забезпечують її ефективну, енергоощадну та безпечну роботу в умовах автономного теплопостачання. Ці вимоги визначають основні властивості системи, її структуру та критерії оцінки працездатності.

Функціональні вимоги описують перелік дій, які система повинна виконувати для гарантування стабільного контролю та автоматизованого керування теплоенергетичними процесами.

Система контролю повинна вимірювати температуру в ключових точках гідравлічної схеми, зокрема на вході та виході сонячного колектора, у баку-накопичувачі та басейні, з використанням цифрових датчиків DS18B20 з точністю не гірше ±0,5 °C, здійснювати аналіз температурних параметрів у реальному часі шляхом порівняння показників Твх, Твих, температури в баку та басейні з метою визначення оптимальних режимів циркуляції теплоносія, автоматично керувати насосами та реле на основі значень температурних різниць ΔT1 між входом і виходом колектора та ΔT2 між баком-накопичувачем і басейном із застосуванням алгоритму з гістерезисом і захистом від частого перемикання.

Також система повинна здійснювати контроль витрати теплоносія за допомогою витратоміра YF-S201 для визначення фактичної роботи насосів, виявлення блокування контуру, відсутності циркуляції або зниження ефективності обігріву, відображати інформацію про роботу системи на двох дисплеях, а саме локальному TFT-дисплеї ILI9341 для детального моніторингу та налаштувань і OLED-дисплеї SSD1306 для дублювання основних параметрів,

передавати дані на віддалений модуль через канал радіозв’язку NRF24L01, забезпечуючи двосторонній обмін між контролерами STM32F411CEU6 і STM32F103C8T6.

Система має підтримувати ручний та автоматичний режими керування з можливістю примусового увімкнення обладнання користувачем або передавання керування алгоритму, реалізовувати захист від перегріву, аварій та помилок, зокрема обриву або некоректної роботи датчиків, відсутності протоку теплоносія, відмови реле чи перевантаження насосів, забезпечувати автоматичне скидання перегрітої води у другий (аварійний) контур шляхом відкриття електрокерованого клапана при перевищенні критичної температури в баку-накопичувачі (зазвичай 85–90 °C), а також зберігати користувацькі налаштування, включаючи порогові температури, режими роботи, гістерезис і інтервали реагування, в енергонезалежній пам’яті мікроконтролера.

Технічні вимоги визначають апаратну структуру системи, її елементну базу, інтерфейси та режими роботи:

- Перший мікроконтролер STM32F411CEU6 повинен здійснювати збір даних, аналіз температурних і витратних характеристик, формувати керуючі сигнали та взаємодіяти з інтерфейсами SPI (дисплей, радіомодуль, мікросхема пам’яті), GPIO (кнопки). Параметри: 512 КБ Flash, 128 КБ RAM, тактова частота до 100 МГц.
- Другий мікроконтролер STM32F103C8T6 повинен приймати команди по радіоканалу (SPI), керувати реле (GPIO), отримувати (USART) та надсилати (SPI) дані з датчиків до головного модуля. А також – відображати основну інформацію на дисплеї SD1306 (I2C).
- Датчики температури DS18B20 повинні підключатися за інтерфейсом OneWire з можливістю паралельного під’єднання додаткових датчиків.
- TFT-дисплей ILI9341 повинен використовувати інтерфейс SPI, забезпечувати кольорову індикацію параметрів та зручний графічний інтерфейс.
- Модулі реле 5В повинні витримувати комутаційний струм не менше 10А та мати оптопару для гальванічної розв’язки.

- Радіомодуль NRF24L01 повинен забезпечувати стабільний обмін даними на відстані не менше 20м у побутових умовах, підтримуючи багатоканальний режим і низьке енергоспоживання.

- Аварійний контур тепловідведення повинен містити: електромагнітний клапан аварійного скидання; окремий канал керування; алгоритм контролю температури; функцію автоматичного повернення до штатного режиму після охолодження.

- Система повинна забезпечувати роботу в температурному діапазоні від 0 до +50 °C, із захистом від стрибків напруги та коротких замикань.

- Усі силові та сигнальні ланцюги повинні мати гальванічну розв’язку, щоб уникнути пошкодження мікроконтролера.

Економічні вимоги визначають доцільність застосування даної системи, її доступність та тривалу ефективність:

- Оптимальне співвідношення “вартість–ефективність” повинно досягатися шляхом використання доступних компонентів високої надійності.

- Енергоефективність системи повинна забезпечувати мінімальне енергоспоживання мікроконтролерів, дисплеїв та радіомодулів.

- Низькі витрати на обслуговування: цифрові датчики DS18B20 не вимагають калібрування; уніфіковані модулі реле полегшують заміну; модульна конструкція спрощує ремонт.

- Довговічність електронних компонентів повинна бути не менше 10 років експлуатації.

- Можливість масштабування і модернізації має передбачати під’єднання додаткових датчиків, дисплеїв, нових модулів керування або альтернативних каналів зв’язку.

- Зниження витрат на підігрів води на 25–40 % має досягатися завдяки автоматичному оптимізованому використанню енергії сонця.

- Використання доступних комплектуючих, наявних на українському ринку, повинно мінімізувати залежність від імпортних поставок.

- Енергоощадність аварійного контуру — використання клапанів та реле з низьким енергоспоживанням і високою механічною надійністю.

Сформульовані функціональні, технічні та економічні вимоги створюють основу для побудови надійної, точної, енергоефективної та безпечної системи контролю сонячного колектора, яка забезпечить стабільний нагрів води, захист від аварійних режимів та комфортну експлуатацію для користувача.

2.3. Розробка структурної схеми системи контролю та управління сонячним колектором

2.3.1. Структурна схема системи контролю.

Для забезпечення автоматичного керування процесом нагріву води в басейні за допомогою сонячного колектора було розроблено структурну схему системи контролю (рис. 2.1).

Система виконує повний цикл автоматизації: вимірювання - аналіз - ухвалення рішення - керування - візуалізація - передача даних - аварійний захист.

Основним обчислювальним елементом є STM32F411CEU6, який приймає інформацію з датчиків (безпосередньо або через радіомодуль), аналізує режими роботи, керує насосами та аварійним клапаном, а також відображає інформацію на TFT-дисплеї.

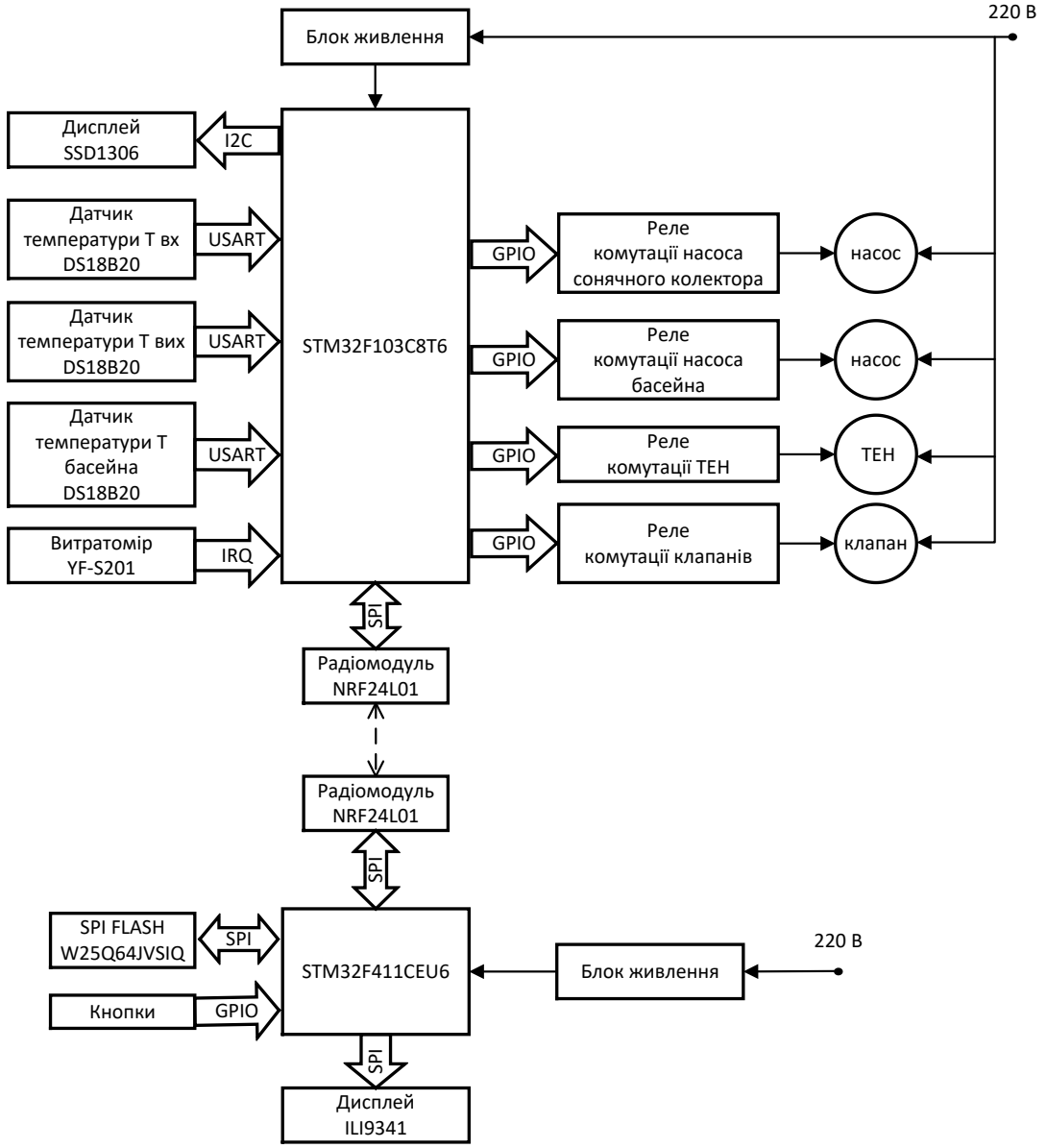

Рис. 2.1. Схема автоматичного керування процесом нагріву води в басейні за допомогою сонячного колектора

Структурна схема містить такі блоки:

- Блок вимірювання температури (DS18B20) – датчики на колекторі, у баку-накопичувачі та в басейні.
- Обчислювально-керуючий блок №1 (STM32F411) – “головний” мікроконтролер.
- Обчислювально-керуючий блок №2 (STM32F103) – віддалений модуль із власним датчиком/датчиками, реле та радіомодулем.

- Блок індикації та візуалізації – TFT-дисплей ILI9341.
- Виконавчий блок: реле керування насосом колектора; реле керування насосом басейну; реле керування ТЕН; реле/клапан аварійного скидання.
- Блок бездротового зв’язку (NRF24L01 або аналог) – обмін між модулями STM32.
- Блок живлення – стабілізоване джерело 5–12 В + опціональне резервне живлення.

Опис структурної взаємодії блоків Між окремими функціональними модулями системи реалізовано чітку логічну та інформаційну взаємодію, що забезпечує послідовний збір вимірювальних даних, їх обробку та формування керуючих впливів відповідно до заданого алгоритму роботи.

Блок вимірювання температури (DS18B20). У системі використовується група цифрових датчиків DS18B20, які вимірюють температуру на виході сонячного колектора, у баку-накопичувачі, у воді басейну та, за потреби, у додатковому аварійному контурі. Датчики передають дані безпосередньо на мікроконтролер STM32F103, який надсилає їх головному контролеру через радіомодуль. Передавання здійснюється по шині OneWire, що дозволяє підключати декілька датчиків на одній лінії.

Обчислювально-керуючий блок №1 (STM32F411) (рис. 2.2) є центральним модулем системи та виконує такі основні функції: приймання даних від віддаленого модуля STM32F103, розрахунок різниць температур ΔT1 та ΔT2, де ΔT1 — різниця між входом і виходом колектора, а ΔT2 — різниця між колектором і басейном, ухвалення рішень щодо вмикання насоса колектора, насоса басейну, аварійного клапана та ТЕН, виявлення перегріву й аварійних ситуацій, керування реле та формування керуючих сигналів, відображення параметрів на дисплеї, зокрема температур, графіків, режимів і станів насосів, обмін інформацією з віддаленим модулем, а також збереження налаштувань користувача в енергонезалежній пам’яті.

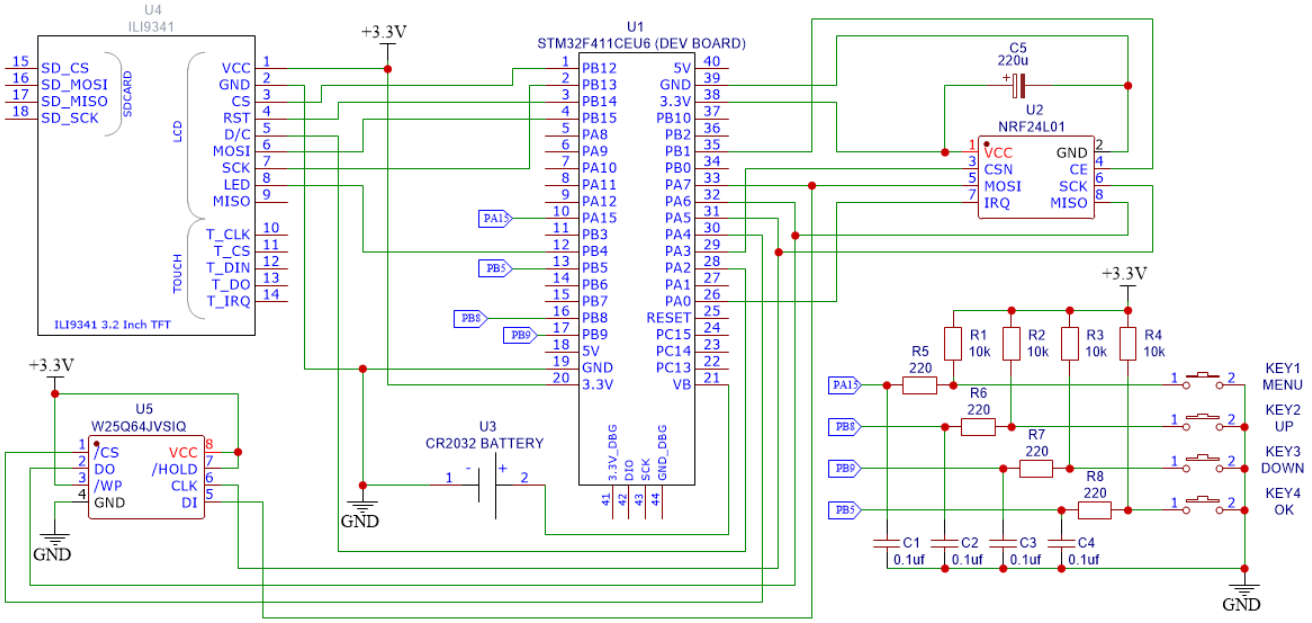

Рис. 2.2. Електросхема блок №1

Програмна логіка побудована за схемою циклічного опитування:

отримання даних - аналіз - керуюча дія - візуалізація - передача даних.

Обчислювально-керуючий блок №2 (STM32F103) (рис. 2.3) виконує такі функції:

- збір локальних температурних даних;
- керування реле на віддаленій частині системи;
- передача даних на головний модуль через бездротовий канал;
- у разі втрати зв’язку з головним модулем на тривалий час (3 хв.) - перехід у безпечний режим (вимкнення реле).

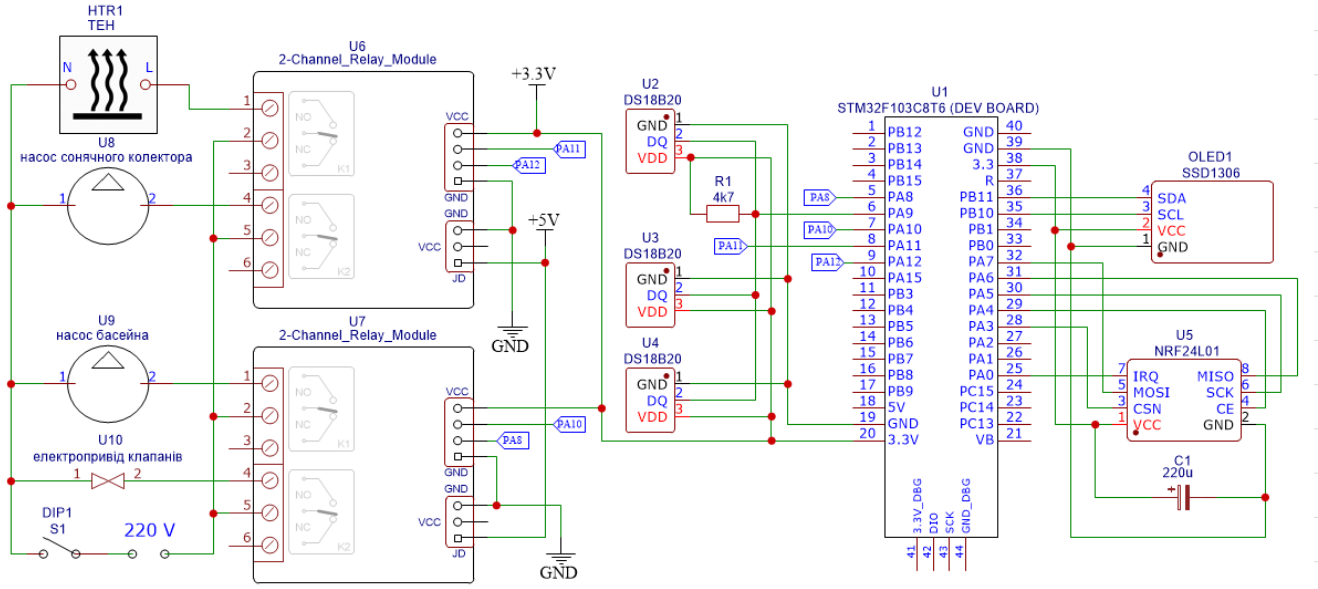

Рис. 2.3. Електросхема блок №2

Цей блок забезпечує децентралізацію та гнучкість розташування компонентів системи.

Блок індикації та візуалізації (TFT ILI9341) забезпечує:

- відображення вибраних режимів роботи;
- відображення значень у режимі реального часу (температур, витрати)
- стан роботи пристроїв: насосів, реле, клапану, ТЕН, радіомодуля;
- відображення графіків температур, витрати, отриманої енергії;
- показ аварійних сигналів (перегрів, відсутність/непрацездатність датчика, помилка зв’язку);
- доступ до меню налаштувань (порогові температури, режими роботи).

Графічний інтерфейс покращує зручність експлуатації.

Виконавчий блок (реле та клапани) складається з трьох основних елементів:

- Реле насоса сонячного колектора – увімкнення, коли ΔT1 > встановленого порогу.
- Реле насоса басейну – увімкнення, коли ΔT2 > параметра користувача.

- Клапан аварійного контуру здійснює автоматичне відкриття при досягненні критичної температури (85–90 °C), а також виконує скидання перегрітої води в резервний контур.

Усі реле мають транзисторні драйвери та гальванічну розв’язку.

- Блок бездротового зв’язку.

Для взаємодії між контролерами використовується радіомодуль NRF24L01. Він виконує функції з передачі даних на відстань ≥ 20 м; має мінімальну затримку сигналу та можливість віддаленого моніторингу.

Блок живлення. Система працює від стабілізованого джерела 5–12 В; опціонально — від акумулятора; через контролер заряду сонячної панелі. Забезпечує захисти від перенапруги, короткого замикання та перевантаження.

2.3.2. Принцип роботи системи Принцип роботи розробленої системи базується на послідовному виконанні етапів ініціалізації, збору вимірювальної інформації, її обробки та формування керуючих впливів відповідно до поточного теплового стану об’єкта.

Після подачі напруги система ініціалізує всі модулі, зокрема датчики, дисплей, реле та модуль зв’язку, мікроконтролер STM32F103 зчитує температурні значення та передає дані головному модулю, STM32F411 отримує всі температури та обчислює різниці ΔT1 і ΔT2, при перевищенні різницею ΔT2 заданого порога відбувається увімкнення насоса басейну, при перевищенні різницею ΔT1 порогового значення здійснюється увімкнення насоса колектора, при зниженні різниць температур нижче мінімальних меж відповідні насоси вимикаються, у разі перевищення температурою у баку аварійного порога система відкриває клапан аварійного контуру, вимикає насоси та подає попередження на дисплеї, усі параметри відображаються на дисплеї й передаються радіоканалом, а користувач має можливість змінювати режими, пороги та параметри роботи системи.

Переваги розробленої структури. Розроблена система контролю та управління сонячним колектором має низку важливих переваг, які забезпечують її ефективність, надійність і зручність у використанні.

Переваги розробленої структури полягають у модульності та розширюваності, високій продуктивності та швидкодії, енергоефективній роботі системи, можливості візуалізації та дистанційного моніторингу, високій точності та надійності вимірювань, наявності аварійного захисту, а також підвищеній безпеці експлуатації.

Розроблена система має гнучку модульну архітектуру, що дозволяє без ускладнень нарощувати її функціональні можливості шляхом інтеграції додаткових датчиків, нових каналів керування або контурів теплообміну. Такий підхід забезпечує адаптивність системи до змінних потреб користувача, модернізації обладнання та умов експлуатації, а також спрощує технічне обслуговування, оскільки окремі елементи можуть бути замінені або оновлені без втручання в роботу всієї системи.

Використання мікроконтролера STM32F411 на базі ядра ARM Cortex-M4 забезпечує значний запас обчислювальної потужності та високу швидкість обробки даних, що дає змогу реалізовувати складні алгоритми контролю й керування, паралельне опрацювання даних від кількох датчиків, фільтрацію вимірювань і компенсацію похибок у режимі реального часу без затримок навіть при розширенні системи.

Енергоефективність системи досягається завдяки оптимізованим алгоритмам керування, за яких насоси вмикаються лише за умови перевищення встановлених температурних порогів, що запобігає зайвій циркуляції теплоносія, зменшує навантаження на обладнання та подовжує термін його експлуатації. Додатково цьому сприяє використання малоспоживальних мікроконтролерів і раціональних режимів роботи периферії.

Система забезпечує наочну візуалізацію параметрів за допомогою TFT-дисплея з контролером ILI9341, на якому відображаються температури, різниці $\Delta T1$ та $\Delta T2$, режими роботи, стани насосів і аварійні повідомлення. Застосування

радіомодуля NRF24L01 дозволяє здійснювати дистанційний моніторинг у реальному часі без необхідності фізичної присутності користувача біля обладнання.

Висока точність і надійність вимірювань забезпечується використанням цифрових датчиків DS18B20 з точністю ±0,5 °C та цифровим інтерфейсом OneWire, що зменшує вплив завад. Реалізовані механізми самодіагностики дозволяють виявляти несправності датчиків і запускати аварійні алгоритми, підвищуючи загальну надійність системи.

Наявність аварійного захисту, зокрема додаткового незалежного контуру для скидання перегрітої води, запобігає небезпечному підвищенню температури та тиску. У разі аварій або відмови датчиків система переходить у безпечний режим, вимикає виконавчі пристрої та інформує користувача через дисплей і радіоканал.

Підвищена безпека експлуатації досягається завдяки гальванічній розв’язці між мікроконтролерами та релейними модулями, наявності резервних режимів роботи й апаратно-програмних захисних механізмів, що гарантує безпечне використання системи в побутових і зовнішніх умовах.

2.4. Вибір елементної бази

Під час розробки системи контролю та управління сонячним колектором важливим етапом є обґрунтований вибір елементної бази, що забезпечує необхідну точність вимірювань, надійність роботи, енергоефективність і простоту інтеграції всіх компонентів.

До складу системи входять: мікроконтролери, датчики температури, блок індикації, виконавчі елементи та модулі зв’язку. Нижче наведено характеристики й обґрунтування вибору кожного елемента.

Мікроконтролер STM32F411CEU6 (рис. 2.4) У якості обчислювального ядра обрано STM32F411CEU6 — високопродуктивний 32-бітний мікроконтролер

на ядрі ARM Cortex-M4, який поєднує потужність, низьке енергоспоживання та широкі можливості периферійного підключення.

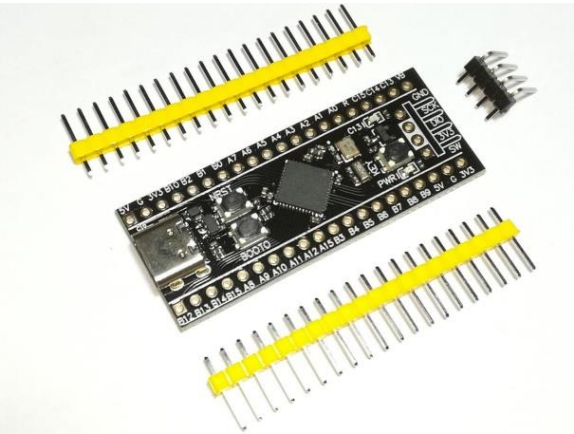

Рис. 2.4. Мікроконтролер STM32F411CEU6

Основні технічні характеристики:

- тактова частота — до 100 МГц;
- об’єм Flash-пам’яті — 512 КБ, оперативної пам’яті — 128 КБ SRAM;
- підтримка периферії: SPI, I²C, UART, ADC, PWM, GPIO;
- наявність апаратного модуля Floating Point Unit (FPU) для обробки математичних операцій;
- робоча напруга — 2,7–3,6 В, споживання енергії менше 100 мкА на 1 МГц тактової частоти.

Мікроконтролер STM32F411CEU6 забезпечує достатню продуктивність для обробки температурних даних, реалізації алгоритмів контролю, керування дисплеєм та взаємодії з модулями зв’язку. Наявність великого набору інтерфейсів дозволяє підключати кілька датчиків і модулів одночасно, що підвищує масштабованість системи.

Мікроконтролер STM32F103C8T6 (Рис. 2.5). обрано для реалізації функцій збирання даних від температурних датчиків, їх первинної обробки та передавання на основний контролер. Це енергоефективна мікросхема сімейства STM32F1 від компанії STMicroelectronics, побудована на ядрі ARM Cortex-M3 з тактовою частотою до 72 МГц.

Мікроконтролер має 64 КБ Flash-пам’яті та 20 КБ SRAM, оснащений інтерфейсами USART, SPI та I²C, 12-бітним АЦП з кількома каналами та таймерами загального призначення. Він працює в діапазоні напруг 2,0…3,6 В і підтримує режими енергозбереження, що є важливим для автономних систем контролю.

Використання STM32F103C8T6 забезпечує достатню обчислювальну продуктивність для опитування датчиків DS18B20, обробки сигналів витратоміра та керування виконавчими механізмами. Наявність стандартних апаратних інтерфейсів спрощує інтеграцію з периферійними модулями та радіоканалом зв’язку.

Обґрунтування вибору мікроконтролера полягає в оптимальному співвідношенні продуктивності та вартості, високій надійності, сумісності з екосистемою STM32 та широкій підтримці бібліотек і прикладів, що спрощує розробку, налагодження та подальшу модернізацію системи.

Рис. 2.5. Мікроконтролер STM32F103C8T6

Датчик температури DS18B20 на рисунку 2.6 та рисунку 2.7. Для вимірювання температури теплоносія використовується цифровий датчик

DS18B20, який підключається до мікроконтролера STM32F103 за інтерфейсом OneWire.

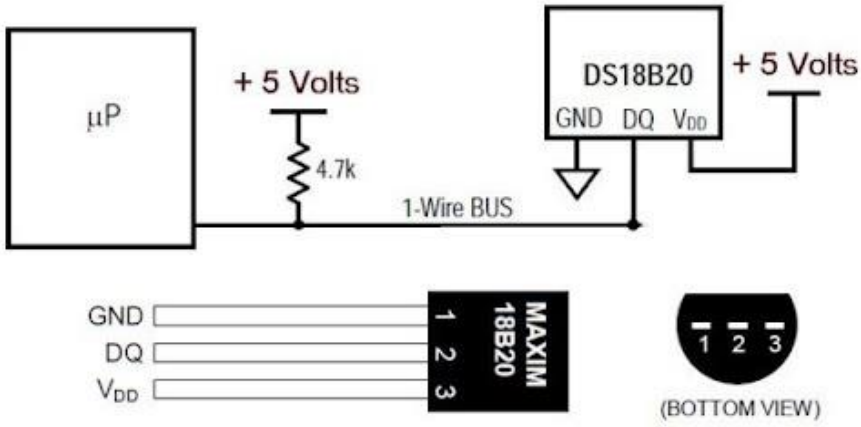

Рис. 2.6. Датчик температури DS18B20

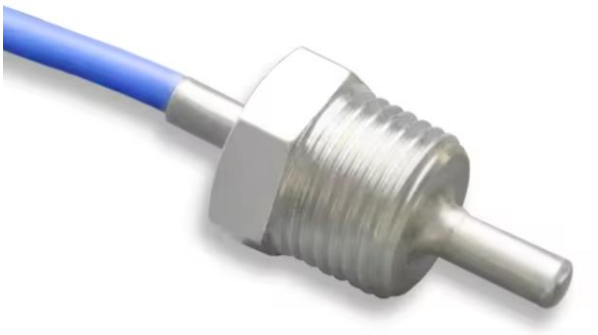

Рис. 2.7. Датчик температури DS18B20 в герметичному виконанні

Основні характеристики:

- діапазон вимірювання: від –55 до +125 °C;
- точність: ±0,5 °C у діапазоні –10...+85 °C;
- роздільна здатність: від 9 до 12 біт (налаштовується програмно);
- можливість підключення кількох датчиків на одній шині OneWire;
- цифровий вихід — не потребує калібрування.

DS18B20 відзначається високою точністю, низьким рівнем шуму та стійкістю до зовнішніх впливів. Його герметичне виконання дозволяє безпосередньо контактувати з рідиною, що важливо для вимірювання

температури води у басейні та в контурі колектора. Крім того, цифровий інтерфейс спрощує підключення до мікроконтролера без необхідності використання аналогових каналів.

Дисплей ILI9341 (TFT 2.4”) зображений на рисунку 2.8. Для візуалізації параметрів системи використовується TFT-дисплей з контролером ILI9341, який має роздільну здатність 320×240 пікселів і підключається до мікроконтролера через інтерфейс SPI.

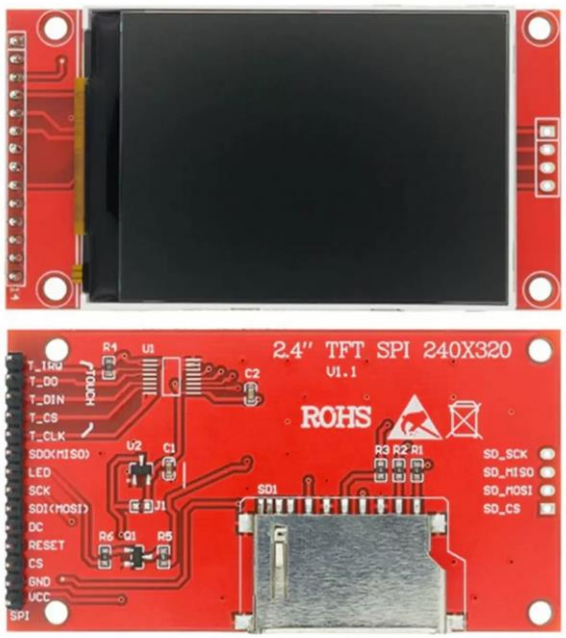

Рис. 2.8. Дисплей ILI9341 (TFT 2.4”)

Основні характеристики:

- тип дисплея: TFT LCD, 16-бітна кольорова палітра;
- інтерфейс підключення: SPI / паралельний;
- робоча напруга: 3,3 В;
- висока контрастність і швидке оновлення зображення;
- підтримка сенсорного шару (опціонально).

Обґрунтування вибору:

Дисплей з ILI9341 забезпечує зручну візуалізацію інформації — температур, стану насоса, режиму роботи та попереджень. Його використання

підвищує ергономічність системи, дозволяє реалізувати просте меню користувача та зменшує потребу у зовнішніх індикаторах.

Реле зображене на рисунку 2.9. Для керування циркуляційним насосом використовується електромеханічне або твердотільне реле, що комутує силове навантаження з мікроконтролера.

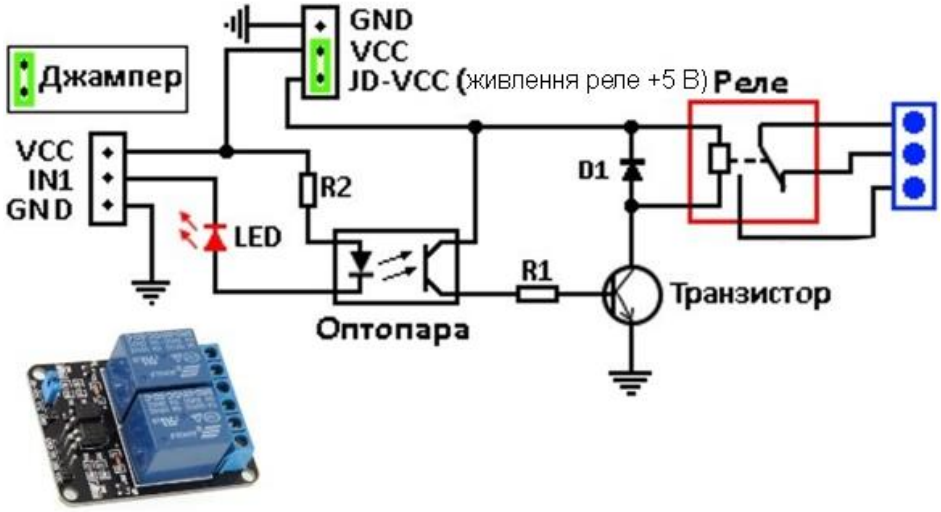

Рис. 2.9. Принципова схема для прикладу одного каналу реле

Основні характеристики:

- робоча напруга котушки: 5 В;
- максимальний струм комутації: 10 А при 220 В;
- гальванічна розв’язка керуючого та силового кола;
- керування сигналом GPIO з мікроконтролера.

Реле дозволяє безпечно керувати насосом, забезпечуючи електричну ізоляцію між логічною частиною системи і силовою мережею. Простота інтеграції та надійність комутації роблять цей тип виконавчого елемента оптимальним для системи контролю.

Радіоприймач/модуль бездротового зв’язку зображене на рисунку 2.10. Для дистанційного моніторингу параметрів системи та обміну даними між мікроконтролерами використовується радіомодуль NRF24L01 або аналогічний пристрій, що працює в діапазоні частот 2,4 ГГц. Застосування бездротового

каналу зв’язку дозволяє передавати виміряні температурні та експлуатаційні параметри без використання додаткових кабельних з’єднань, що підвищує гнучкість і надійність системи.

Модуль підтримує швидкість передавання даних до 2 Мбіт/с, підключається до мікроконтролера за інтерфейсом SPI та характеризується низьким енергоспоживанням. Дальність зв’язку в умовах відкритого простору може досягати 100 м, що є достатнім для задач моніторингу та керування елементами системи сонячного колектора.

Використання NRF24L01 забезпечує стабільний обмін даними між основним і допоміжним контролерами, а також створює можливість подальшого розширення функцій системи, зокрема віддаленого моніторингу та діагностики.

Рис. 2.10. Принципова схема для прикладу одного каналу реле

2.5. Розробка алгоритму роботи системи контролю

Алгоритм роботи системи контролю сонячного колектора забезпечує автоматичне зчитування параметрів, аналіз стану системи та вибір режимів роботи насосного обладнання. На рисунках 2.11–2.15 подано поетапну блок-схему функціонування, яка відображає логіку обробки даних від датчиків і прийняття керуючих рішень мікроконтролером STM32.

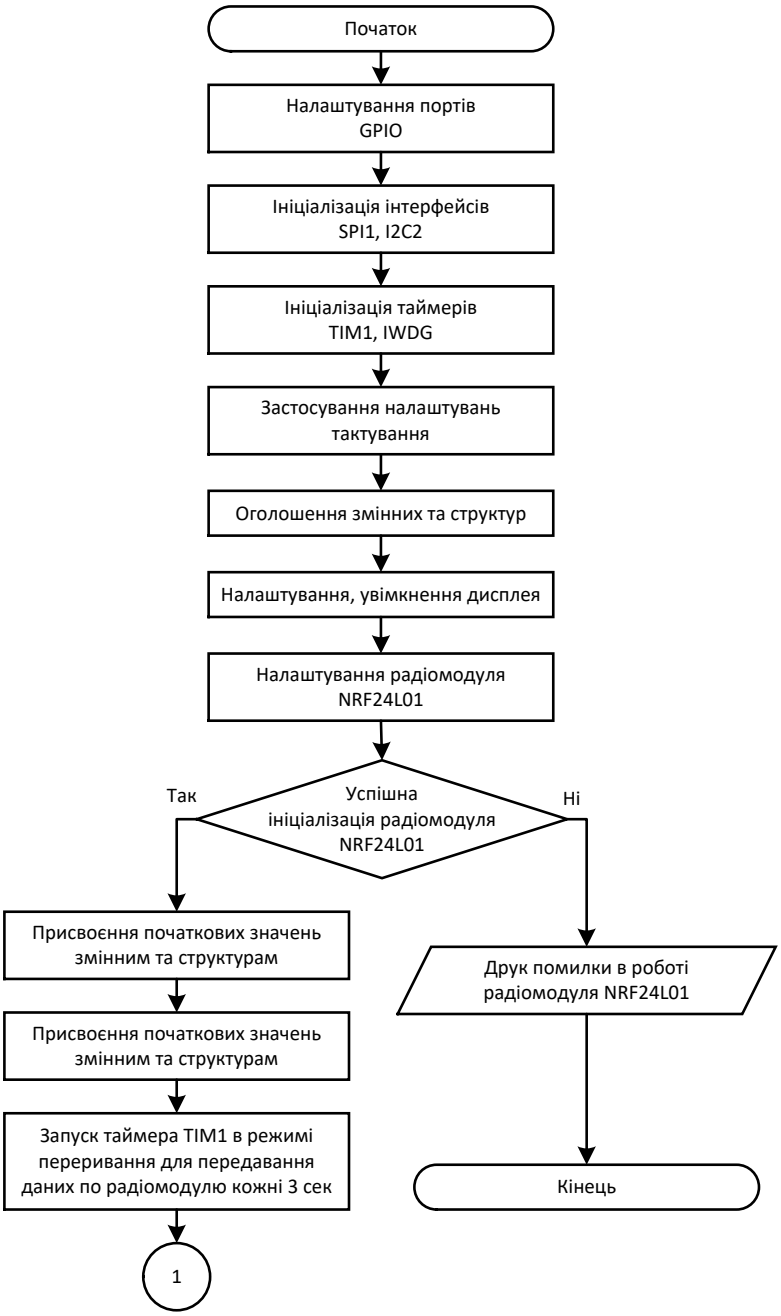

Рис. 2.11. Блок-схеми блока 2

На початковому етапі здійснюється ініціалізація мікроконтролера та апаратних модулів, зокрема периферійних інтерфейсів SPI та I2C, датчиків температури DS18B20, витратоміра та TFT-дисплея ILI9341, а також виконується перевірка працездатності системи. Далі завантажуються збережені налаштування та встановлюються початкові параметри, після чого перевіряється коректність підключення датчиків. У разі їх відсутності система переходить в аварійний режим із відповідним повідомленням користувачу.

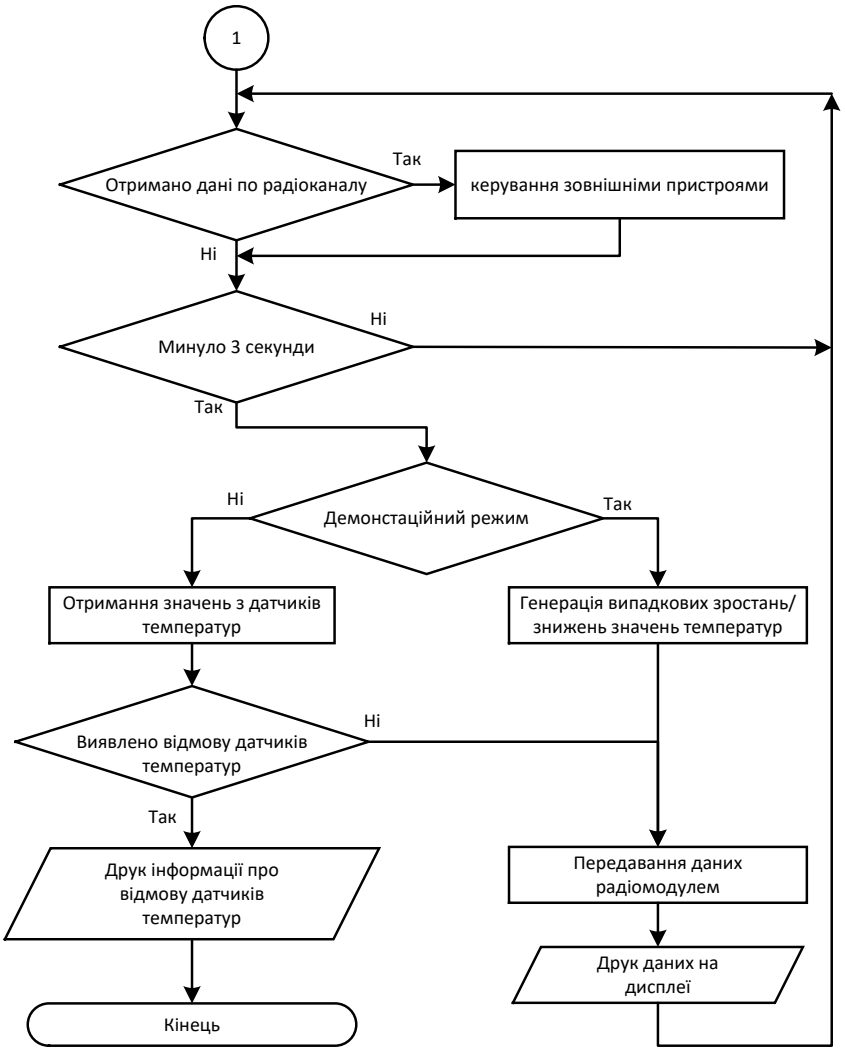

Рис. 2.12. Продовження блок-схеми блока 2

У робочому циклі основною операцією є аналіз температурних різниць ΔT1 та ΔT2, на основі яких формується рішення щодо увімкнення або вимкнення насосів. Увімкнення насоса здійснюється за умови достатньої різниці температур для ефективного нагріву, а вимкнення — у разі відсутності теплової доцільності циркуляції. Після виконання керуючих дій алгоритм повертається до початку циклу та повторює перевірку стану системи.

Рис. 2.13–2.15 демонструють детальну логіку роботи системи при взаємодії з усіма доступними датчиками: температури на вході та виході колектора, температури басейну, температури зовнішнього середовища (за наявності), датчика витрати.

Рис. 2.13 демонструє етап зчитування всіх значень датчиків у циклі. При цьому модуль контролю перевіряє коректність отриманих даних, наявність помилок (обрив датчика, некоректні значення). У разі некоректних значень алгоритм переходить до гілки аварійної обробки — система вимикає насос, надсилає повідомлення та запобігає подальшій роботі в небезпечному стані.

Рис. 2.13. Блок-схема блока 1

Аналіз отриманих показників (рис. 2.14)

На цьому етапі проводиться порівняння значень температури з порогами:

- Тколектора > Тбасейну + ΔT_on - необхідно ввімкнути насос;
- Тколектора < Тбасейну + ΔT_off - необхідно вимкнути насос;
- перевищення критичної температури колектора - аварійне вимкнення.

У разі переходу в аварійний стан відображається відповідне попередження, а система блокує насос до нормалізації параметрів.

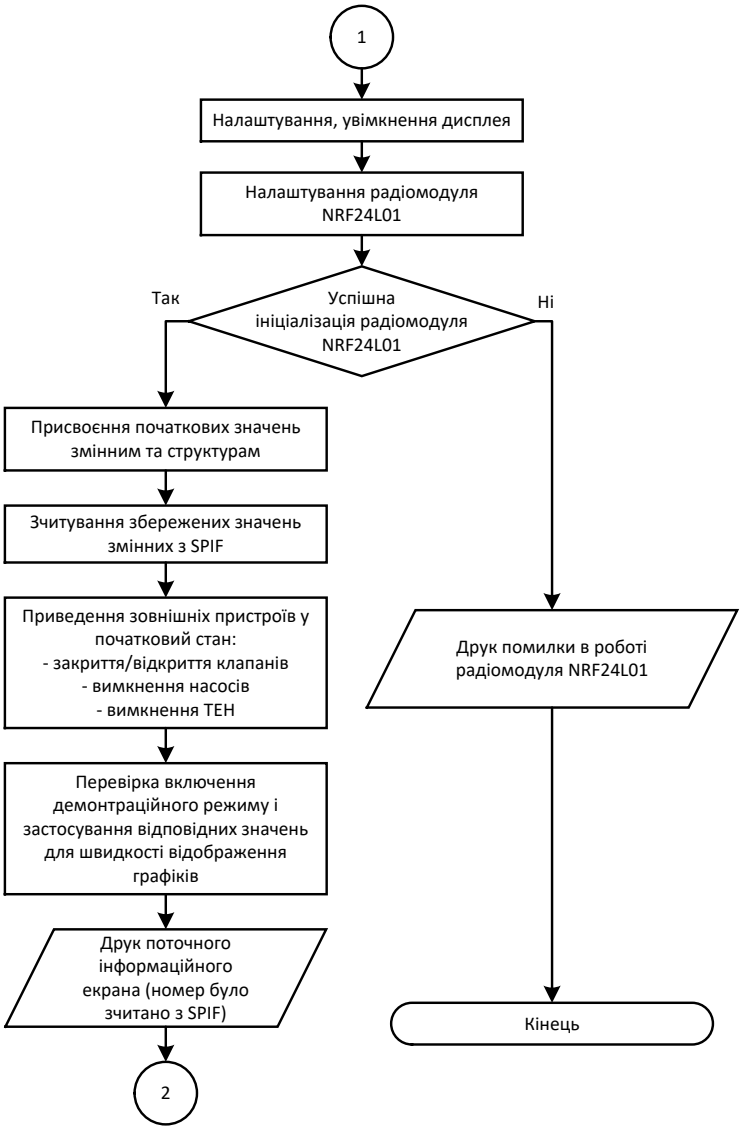

Рис. 2.14. Продовження блок-схеми блока 1

Опрацювання режимів роботи (рис. 2.15) Завершальна частина алгоритму демонструє реалізацію перевірок перегріву колектора та необхідності активувати аварійний контур, досягнення заданої температури басейну та переходу системи в енергозберігаючий режим, перевірку працездатності насоса (через датчик потоку).

Після проходження усіх етапів система повертається до основного циклу обробки даних і продовжує роботу в безперервному режимі.

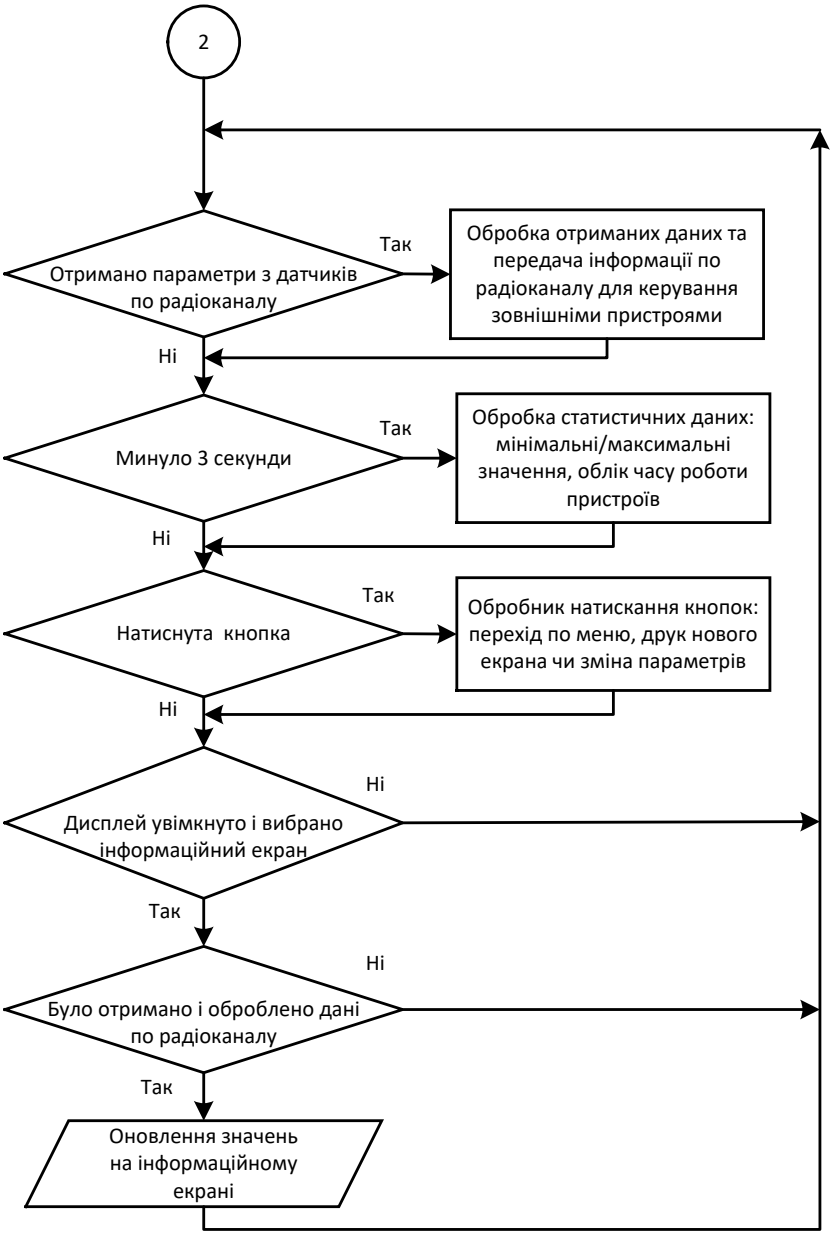

Рис. 2.15. Продовження блок-схеми блока 1

Алгоритм, представлений у блок-схемах, забезпечує стабільну й безпечну роботу сонячного колектора, автоматичне керування насосами на основі аналізу температурних різниць, захист від перегріву та помилок у роботі датчиків, можливість інтеграції з інтерфейсом користувача для налаштувань та моніторингу. Візуально подані блок-схеми відображають загальну структуру роботи системи.

2.6. Висновок до розділу 2

У другому розділі здійснено комплексне теоретико-аналітичне обґрунтування, структурно-функціональне моделювання та алгоритмічне формування системи контролю та управління сонячним колектором, що є ключовою частиною побудови енергоефективної установки для нагріву води. На основі проведеного аналізу визначено необхідність застосування мікропроцесорної платформи з розподіленою архітектурою, що забезпечує підвищену надійність, точність вимірювань і гнучкість у керуванні теплотехнічними процесами.

У межах постановки задачі сформульовано цілі та критерії функціонування системи, до яких належать підтримання стабільного температурного режиму води в басейні, оптимізація теплообмінних процесів між сонячним колектором, баком-накопичувачем і зовнішніми споживачами, реалізація комплексу захисних заходів, а також організація надійного моніторингу параметрів у режимі реального часу. На підставі цього визначено структурні, функціональні, інформаційні, технічні та експлуатаційні вимоги до системи.

Проведений вибір елементної бази обґрунтовує застосування сучасних електронних компонентів з урахуванням їх технічних характеристик, енергоефективності, надійності та відповідності умовам експлуатації. Вибрані мікроконтролери STM32F411CEU6 та STM32F103C8T6 забезпечують розподілену обробку даних, апаратну підтримку периферійних інтерфейсів, можливість реалізації складних алгоритмів керування та достатній обчислювальний ресурс. Комплекс цифрових датчиків DS18B20, витратомір YF-S201, модуль бездротового зв’язку NRF24L01, виконавчі релейні модулі та TFT-дисплей ILI9341 формують технічно збалансовану базу для побудови надійної вимірювально-керуючої системи.

Розроблена структурна схема відображає логічну та інформаційну взаємодію між основними підсистемами: блоком сенсорного моніторингу,

обчислювально-керуючими модулями, блоком індикації, виконавчими пристроями та засобами бездротової передачі даних. Така архітектура забезпечує масштабованість, модульність та підвищену відмовостійкість системи, а також створює умови для подальшого розширення її функціональних можливостей.

На основі структурної моделі сформовано детальний алгоритм функціонування системи, поданий у вигляді блок-схем. Алгоритм охоплює всі етапи роботи — від ініціалізації апаратних модулів і діагностики сенсорів до обробки температури, реалізації логіки керування насосним обладнанням, виконання захисних процедур і забезпечення інформаційної взаємодії між основним та віддаленим контролерами. Особливу увагу приділено впровадженню механізмів захисту від перегріву, контролю працездатності датчиків, перевірки наявності потоку теплоносія та запобігання частому перемиканню реле, що позитивно впливає на ресурс обладнання і підвищує загальну надійність системи.

Таким чином, результати, отримані у розділі 2, формують науково обґрунтовану й технічно цілісну основу для побудови інтелектуальної системи контролю та управління сонячним колектором. Визначені вимоги, розроблені структурні рішення та алгоритми забезпечують передумови для ефективного функціонування системи, підвищення енергоефективності, реалізації захисних механізмів та забезпечення високої точності теплотехнічних параметрів.

Сформована архітектура та алгоритмічне забезпечення створюють необхідні умови для переходу до практичних етапів розробки — проєктування електричних принципових схем, реалізації програмного забезпечення, моделювання режимів роботи та подальшої експериментальної перевірки працездатності системи.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КОНТРОЛЮ

3.1. Вибір середовища програмування

Для розроблення програмного забезпечення системи контролю сонячного колектора обрано середовище STM32CubeIDE, яке є офіційним інтегрованим інструментом розробки компанії STMicroelectronics. Використання цього середовища забезпечує сумісність із мікроконтролерами сімейства STM32 та підтримує основні етапи створення вбудованого програмного забезпечення.

STM32CubeIDE базується на платформі Eclipse та містить засоби для конфігурування апаратних ресурсів, компіляції та налагодження програм. За допомогою STM32CubeMX виконується налаштування тактування, GPIO та периферійних модулів, а вбудований компілятор GCC для ARM Cortex-M забезпечує оптимізацію коду. Вбудований відлагоджувач із підтримкою інтерфейсу SWD дозволяє здійснювати налагодження в режимі реального часу.

У програмному забезпеченні використовується бібліотека HAL, яка спрощує роботу з периферійними модулями мікроконтролера та зменшує час розробки. Бібліотека застосовується для реалізації обміну даними через SPI, 1-Wire, роботи з таймерами та керування виконавчими пристроями системи.

Додатково у процесі розробки використовуються засоби графічного моніторингу змінних у реальному часі та програмні середовища для попередньої перевірки схемних рішень.

Мова програмування — C, що забезпечує ефективне використання ресурсів мікроконтролера та необхідну продуктивність системи.

Таким чином, використання STM32CubeIDE є доцільним для реалізації програмної частини системи контролю сонячного колектора та забезпечує зручність налагодження і стабільність роботи програмного забезпечення.

3.2. Структура програмного забезпечення системи

Система контролю сонячного колектора побудована з двох окремих блоків, кожен з яких базується на своєму МК: STM32F411CEU6 та STM32F103C8T6, які обмінюються інформацією по радіоканалу. Кожен блок має своє програмне забезпечення. Це забезпечує високу гнучкість, зручність налагодження та можливість подальшої модернізації.

Логічна структура програм складається з низки функціональних компонентів, кожен із яких відповідає за окремий етап роботи системи контролю температури води.

Архітектура ПЗ поділяється на такі основні рівні: рівень апаратної абстракції (HAL). Цей рівень включає драйвери та бібліотеки STM32Cube HAL, які забезпечують доступ до периферійних модулів мікроконтролера (GPIO, I2C, UART, SPI, TIM), дозволяють працювати з датчиками та виконавчими елементами незалежно від конкретної апаратної реалізації, а також спрощують ініціалізацію та конфігурування апаратних ресурсів.

Використання HAL-бібліотек підвищує переносимість коду та зменшує обсяг низькорівневих операцій.

Блок на базі МК STM32F103C8T6 відповідає за опитування датчиків температури DS18B20 через інтерфейс 1-Wire, опитування датчика витрати шляхом підрахунку імпульсів обертів, перевірку працездатності датчиків, керування реле управління зовнішніми пристроями, а також двосторонній обмін даними по радіоканалу з іншим блоком.

Основні функції блока включають Get_Temperatures() - отримання температур з датчиків DS18B20, readNRF24() - отримання даних від іншого блока за допомогою радіомодуля NRF24L01, packetToByteArray_TX() - підготовку даних для передавання, nrf24_transmit() - передавання даних, TriangleGenRand_Next() та TriangleGenRand_Next_Delta() - генерування випадкової поведінки зміни значень температур для перевірки працездатності алгоритмів у демонстраційному режимі.

Функція main() реалізує головний цикл програми, координує виконання всіх інших функцій, керує зовнішніми пристроями на основі даних, отриманих від іншого блока, виводить основні значення параметрів на дисплей.

Перевірка працездатності датчиків температури інтегрована в бібліотеку для них (OneWire).

Блок на базі МК STM32F411CEU6 відповідає за збір та аналіз отриманої інформації, збереження історії даних у енергонезалежну пам’ять, відображення параметрів на TFT-дисплеї у текстовому вигляді та у вигляді графіків, взаємодію з користувачем через кнопковий інтерфейс, застосування програмної фільтрації (ковзне середнє), прийняття рішень щодо керування зовнішніми пристроями залежно від вибраних режимів та поточних значень параметрів, а також двосторонній обмін даними по радіоканалу з іншим блоком.

Основні функції блока включають button_click_handler() - обробник натискання кнопок, printInfoStatic() та printInfoUpdate() - друк і оновлення інформаційних екранів, printChart() та updateChart() - друк і оновлення графіків, writeSPIFlash() і readSPIFlash() - запис та читання даних з енергонезалежної пам’яті, readNRF24() - отримання даних від іншого блока за допомогою радіомодуля NRF24L01, CheckON() - перевірку статусу системи з метою визначення необхідності увімкнення або вимкнення зовнішніх пристроїв.

Основним модулем прийняття рішень є функція CheckON(). У ній реалізовано алгоритм вмикання та вимикання зовнішніх пристроїв (насоси, клапан, ТЕН), захист від замороження шляхом вмикання ТЕН при наближенні температури до 0 °C, захист від перегріву колектора з аварійним скиданням гарячої води в додатковий бак при досягненні встановленої температури (70–90 °C), а також оптимізацію перемикань з метою економії енергії.

Для роботи насоса басейна реалізовано наступні режими підтримання температури в басейні:

- Режим 1: Спортивне плавання: 24 – 26 °C;
- Режим 2: Купання дорослих: 26 - 28 °C;
- Режим 3: Діти, сімейне купання: 28 - 30 °C;

- Режим 4: Терапевтичні цілі: 30 - 32 °C;
- Режим 5: Користувацький режим;
- Режим 6: Насос басейна вимкнено.

В користувача є можливість змінювати значення температур в кожному з режимів.

Для насоса сонячного колектора реалізовано наступні режими роботи:

- Режим 1: Насос вимкнено;
- Режим 2: Циркуляція при $\Delta T > 0$ °C;
- Режим 3: Циркуляція при $\Delta T > 8$ °C.

Режим 3 є найбільш економічним та ефективним. Значення ΔT для нього користувач може відкоригувати під свої умови.

Для роботи ТЕН створено наступні режими:

- Режим 1: ТЕН вимкнено;
- Режим 2: ТЕН підтримує T = 33 °C в часові інтервали;
- Режим 3: ТЕН підтримує T = 33 °C постійно.

Для режиму 2 користувач може налаштувати таблицю роботи часових інтервалів. Для підтримання необхідної температури, користувач може змінити ΔT.

Математична модель також включає обчислення отриманої сонячної енергії. Для визначення кількості отриманої енергії від сонячного колектора за час Δt застосовується формула 3.1:

$$\Delta Q = c \cdot \rho \cdot V \cdot \Delta T \cdot \Delta t, \quad (3.1)$$

де ΔQ — кількість отриманої теплової енергії від сонячного колектора за час Δt (в Джоулях) або кВт·год: 1 кВт = 1 Дж / 3 600 000

c — питома теплоємність води (≈ 4186 Дж/(кг·°C))

ρ — густина води (≈ 1000 кг/м³)

V — об’ємна витрата води (м³/с)

ΔT — різниця температур (°C)

Δt — час протікання (с)

Функції інтерфейсу користувача (printInfoStatic(), printInfoUpdate(), printChart(), updateChart()) відповідають за відображення даних на TFT-дисплеї у текстовому вигляді та у вигляді графіків, відображення вибраних режимів роботи та поточного стану зовнішніх пристроїв, інформування про аварійні події (перегрів, помилка датчиків). За допомогою цих функцій реалізовано інформаційні екрани, меню налаштувань та аварійні повідомлення. Функція main() виконує ті ж задачі, що і в іншому блоці.

Отже, структура програмного забезпечення побудована так, щоб забезпечити стабільну роботу системи, розділити функції на незалежні модулі, полегшити тестування та модернізацію, а також забезпечити масштабованість відповідно до потреб користувача.

3.3. Опис основних функцій програми

Функціонування системи контролю сонячного колектора базується на наборі ключових програмних функцій, які виконують приймання та обробку параметрів, керування зовнішніми пристроями та оновлення інтерфейсу користувача. У цьому підпункті подано опис основних функцій, що реалізують роботу системи на рівні програмного коду.

Функції зчитування та обробки даних датчиків включають get_ROMid(), яка використовується модулем на базі STM32F103C8T6 для ініціалізації цифрових датчиків температури DS18B20 та отримання їхніх адрес, а також Get_Temperature(), що застосовується модулем на базі STM32F103C8T6 для опитування цифрових датчиків температури DS18B20 та забезпечує зчитування температури кожного датчика і виконання базової перевірки коректності даних, зокрема виявлення обриву та помилкових значень.

readNRF24() – Функція виконує приймання радіоданих через модуль NRF24L01 (Рисунок 3.1). Вона:

- читає вхідний пакет у буфер dataRX;

- аналізує статус приймання;
- визначає наявність помилок.

Це основний канал надходження даних у канал керування.

```
void readNRF24(void) //отримання даних з радіомодуля NRF24L01
{
    irq_RX = 0;
    if(nrf24_data_available())
    {
        uint8_t status = nrf24_r_status();

        if (status & (1 << RX_DR))  // Дані отримані успішно
        {
            nrf24_receive(dataRX, PLD_SIZE);
            BufferToPacket_RX(dataRX, &packet_RX);
            Flags = packet_RX.flags;
            error_RX = 0;       //скидаємо кількість невдалих отримань

            //отримали нові дані, можливо потрібно змінити стан пристроїв:
            PUMP_APPLY_STATE();
            HEAT_APPLY_STATE();
            PUMPPOOL_APPLY_STATE();
        }
        else
        { // Помилка при отриманні
#if DEBUG
            printf("ERROR RX!\n");
#endif
            nrf24_clear_rx_dr(); // Очистити прапорець
            nrf24_flush_rx(); // Очистити буфер

            nrf24_stop_listen();
            HAL_Delay(300);
            nrf24_listen();
        }
    }
}
```

Рис. 3.1. Лістинг функції readNRF24()

Функції BufferToPacket_RX() та packetToByteArray_TX() відповідають за перетворення масиву байтів у структуру з температурою та витратою, а також за зворотне перетворення структури у байтовий пакет для передавання, що забезпечує узгодженість формату обміну даними між двома мікроконтролерами.

Функція packetToByteArray_TX() (Рис. 3.2), переводить значення t1, t2, t3 та V із типу int16_t у вісім окремих байтів, гарантує правильний порядок байтів (little-endian) та формує масив dataTX[8], який передається через функцію nrf24_transmit().

```
void packetToByteArray_TX(volatile Packet_TX *packet0, uint8_t *byteArray) {
    // Записуємо температуру (2 байти)
    byteArray[0] = (uint8_t)(packet0->t1 & 0xFF);        // молодший байт
    byteArray[1] = (uint8_t)((packet0->t1 >> 8) & 0xFF); // старший байт

    // Записуємо температуру (2 байти)
    byteArray[2] = (uint8_t)(packet0->t2 & 0xFF);        // молодший байт
    byteArray[3] = (uint8_t)((packet0->t2 >> 8) & 0xFF); // старший байт

    // Записуємо температуру (2 байти)
    byteArray[4] = (uint8_t)(packet0->t3 & 0xFF);        // молодший байт
    byteArray[5] = (uint8_t)((packet0->t3 >> 8) & 0xFF); // старший байт

    // Записуємо кількість імпульсів датчика витрати (2 байти)
    byteArray[6] = (uint8_t)(packet0->V & 0xFF);         // молодший байт
    byteArray[7] = (uint8_t)((packet0->V >> 8) & 0xFF);  // старший байт
}
```

Рис. 3.2. Лістинг функції packetToByteArray_TX()

Функція BufferToPacket_RX() (Рис. 3.3), копіює весь буфер у структуру Packet_RX та забезпечує швидке перетворення вхідного пакета у внутрішній формат. Структура Packet_RX містить flags — значення стану пристроїв, а також payload — масив байтів для заповнення решти структури з метою вирівнювання розмірів пакетів під час передавання та приймання.

```
static inline void BufferToPacket_RX(const uint8_t *buffer, Packet_RX *out_pkt) {
    memcpy(out_pkt, buffer, PLD_SIZE);
}
```

Рис. 3.3. Лістинг функції BufferToPacket_RX()

Функція модуля на базі STM32F103 HAL_TIM_PeriodElapsedCallback() (рис. 3.4), викликається кожні 3 секунди та відповідає за оновлення значень витрати води, синхронізацію циклів зчитування та синхронізацію радіообміну.

Функції модуля на базі STM32F411 HAL_TIM_PeriodElapsedCallback() та ToDo_every_3sec() (Додаток Б), також викликаються кожні 3 секунди та відповідають за синхронізацію радіообміну, запис нових точок у графік, облік часу роботи пристроїв і збір статистики мінімальних та максимальних значень параметрів. Це є ключовим механізмом підтримки стабільної часової роботи системи.

```
  //обробник переривань таймера (раз на 3 секунди)
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
#if  !DEBUG
    HAL_IWDG_Refresh(&hiwdg); // Скидання сторожового таймера
#endif

    if(htim->Instance == TIM1) //перевіряємо чи переривання від 1-го таймера TIM1
    {
        irq_3sec = 1;
        error_RX++;
    }
}
```

Рис. 3.4. Лістинг функції HAL_TIM_PeriodElapsedCallback()

Опитування стану системи здійснюється після ініціалізації всіх пристроїв та глобальних змінних в обох блоках, після чого встановлюється відповідна ознака system_ready = 1. До цього моменту функції та обробники переривань не виконують жодних дій, що забезпечує захист від передчасного запуску системи.

Оновлення станів реле (насоси, ТЕН, клапан) у модулі на базі STM32F103 виконується кожні 3 секунди шляхом приведення фізичних реле у відповідність до значень прапорців у структурі Flags у безкінечному циклі за ознакою переривання irq_3sec == 1, з використанням функцій PUMP_APPLY_STATE() — для насоса сонячного колектора, HEAT_APPLY_STATE() — для ТЕН, PUMPPOOL_APPLY_STATE() — для насоса басейна, VALVE_APPLY_STATE() — для аварійного клапана. Це гарантує, що стан виконавчих елементів завжди синхронізований з останньою керуючою командою.

CheckON() — головна логічна функція системи в модулі на базі STM32F411 (Додаток Б), яка відповідає за прийняття рішень щодо вмикання та вимкнення насоса сонячного колектора, керування насосом басейна згідно з режимом, вибраним користувачем, захисту від замерзання шляхом вмикання ТЕН при низькій температурі, захисту від перегріву з увімкненням аварійного клапана, оптимізації частоти перемикань із забезпеченням мінімального часу роботи та простою, а також реакції на помилки зчитування датчиків. Ця функція є центральною частиною алгоритмічного ядра програми.

Функції інтерфейсу користувача printInfoStaticX(), printInfoUpdateX() та printChart3() у модулі на базі STM32F411 відповідають за відображення та оновлення інформаційних параметрів у режимі реального часу, зокрема показ загальної та останніх тривалостей роботи і простою пристроїв, відображення статистичних даних у вигляді максимальних і мінімальних значень параметрів, поточних значень параметрів, вибраних режимів, стану роботи пристроїв, а також друк графіків (Додаток Б).

Функція button_click_handler() реалізує обробку натискання кнопок. Після кожного натискання виконується скидання таймера активності дисплея, що забезпечує автоматичне вимикання підсвічування при тривалій бездіяльності (Додаток Б). У межах функції здійснюється зчитування бітової маски натиснутих кнопок через KEYB_Inkeys(), визначення типу натискання (вгору, вниз, вліво, вправо, вибір, повернення), перемикання пунктів меню та інформаційних екранів, зміна значень параметрів налаштувань (температури, гістерезису, режимів роботи насосів і ТЕН), а також формування ознаки переходу на новий екран.

Функція повертає значення 1 у разі зміни екрана або контексту меню та значення 0, якщо обробка натискання не потребує зміни екрана. На основі цього значення головний цикл програми викликає функцію Goto_New_Window(), яка відображає відповідний інтерфейс.

Функція Goto_New_Window() в модулі на базі STM32F411 (Рис. 3.5).

Ця функція відповідає за друк інтерфейсу користувача при переході на новий екран. Вона викликається після обробки кнопок у разі зміни активного пункту меню або інформаційної сторінки.

Функція забезпечує чіткий поділ між незмінною частиною інтерфейсу (статичний вміст) та даними, що оновлюються в реальному часі.

```
//функція відображає новий інфо екран
void Goto_New_Window(void)
{
    if(ScreenMode->Get_Poz(ScreenMode)  ==  0)  //якщо на одному з інфо екранів
    {
        LCD_Fill(lcd, fm->color_scheme->bg_color);
        switch (Screens[0]->Get_Poz(Screens[0]))    // в залежності від номера екрана
        {
            case 0: // інфо екран 0
                    printInfoStatic0(); // Друк статичної картинки екрана 0
                    printInfoUpdate0(); // Друк значень на екрані 0
                    break;
            case 1: // інфо екран 1
                    printInfoStatic1(); // Друк статичної картинки екрана 1
                    printInfoUpdate1(); // Друк значень на екрані 1
                    break;
            case 2: // інфо екран 2
                    printInfoStatic2();     // Друк статичної картинки екрана 2
                    printInfoUpdate2(); // Друк значень на екрані 2
                    break;
            case 3: // інфо екран 3
                    printChart3(); // Друк графіків
        }
    }
}
```

Рис. 3.5. Лістинг функції Goto_New_Window()

Функції роботи з енергонезалежною пам’яттю реалізують керування зовнішньою SPI Flash пам’яттю через високорівневий модуль SPIF, який забезпечує автоматичне керування секторами пам’яті, запис та читання структур налаштувань, збереження графіків температури, а також ведення метаданих для контролю валідності записів.

Головна функція програми в модулі на базі STM32F411 - main() - реалізує повний запуск та керування системою (Додаток Б). Основними етапами її роботи є ініціалізація апаратних функцій, зокрема GPIO, SPI1 та SPI2, ADC, RTC, TIM2 і TIM3, TFT-дисплея ILI9341 та watchdog IWDG, завантаження налаштувань і графіків із зовнішньої SPI Flash за допомогою функції readSPIFlash(), ініціалізація

та налаштування радіомодуля NRF24L01 із встановленням адреси, конфігурацією каналу та переходом у режим очікування приймання.

Після цього виводиться стартова інформація за допомогою функції printStartInfo(), після чого викликається функція Goto_New_Window() для формування першого інтерфейсного вікна.

У головному циклі while(1) постійно виконуються приймання даних з другого модуля за допомогою функції readNRF24(), алгоритм керування пристроями CheckON(), обробка натискання кнопок button_click_handler(), оновлення екрана через Goto_New_Window(), оновлення графіків, збереження даних у SPI Flash за необхідності, а також автоматичне вимикання дисплея.

3.4. Розробка алгоритмів автоматичного регулювання температури

Алгоритми автоматичного регулювання температури забезпечують стабільну роботу системи підігріву води, оптимальне використання сонячної енергії та захист обладнання від небезпечних станів. Логіка функціонування ґрунтується на аналізі вимірювальних даних, динаміці теплових процесів та налаштуваннях, визначених користувачем.

3.4.1. Вхідні дані та їх обробка. Для коректної роботи система потребує актуальних параметрів, що характеризують тепловий стан колектора й басейну. Дані надходять періодично та формують основу для подальших керуючих рішень. Кожен пакет містить числову інформацію про температурні показники та витрату теплоносія, а також службові ознаки, необхідні для контролю достовірності вимірювань.

До складу таких даних належать:

- температура на вході теплообмінника (Твх);
- температура на виході теплообмінника (Твих);
- температура води в басейні (Тбас);
- значення витрати теплоносія (V);

- індикатори коректності роботи датчиків.

Після отримання всі параметри проходять перевірку та попередню обробку. Коректні величини усереднюються за допомогою ковзного середього, що дозволяє зменшити вплив випадкових коливань та підвищує стабільність регуляторних рішень. Саме згладжені дані використовуються як основа для подальшої логіки регулювання.

3.4.2. Контроль коректності даних та загальний алгоритм керування циркуляцією. Перед виконанням основних дій система аналізує достовірність даних. У разі помилок або відсутності зв’язку активується безпечний режим, що передбачає вимкнення циркуляції, нагрівача та закриття запобіжного клапана. Повернення до нормальної роботи можливе лише після відновлення стабільних вимірювань.

Після підтвердження коректності даних здійснюється керування насосом сонячного колектора. Основним критерієм є різниця температур яка обчислюється за формулою 3.2:

$$dT = Твх – Твих. \quad (3.2)$$

Цей параметр відображає ефективність відбору тепла. Насос вмикається при перевищенні встановленого порогу, конкретне значення якого залежить від вибраного користувачем режиму. Після запуску циркуляція повинна працювати мінімальний заданий час, що запобігає надмірній кількості перемикань. Вимкнення можливе лише при зниженні різниці температур на величину гістерезису.

Об’єднання етапів контролю даних та керування циркуляцією формує узгоджений і безпечний механізм роботи сонячного контуру.

3.4.3. Алгоритми підтримання температури басейну та керування електричним нагрівачем. Температура води в басейні підтримується згідно з вибраним режимом, у межах якого встановлені значення Tmin та Tmax. Керування ґрунтується на двопозиційному принципі з гістерезисом: насос вмикається при надмірному охолодженні води і вимикається при її перегріванні. Таке регулювання забезпечує стійкість температури та мінімізує частоту перемикань. Один із режимів передбачає заборону роботи насоса незалежно від температури.

Електричний нагрівач працює у режимах, визначених користувачем: вимкненому, обмеженому часовими інтервалами або постійному. Додатково реалізовано захист від замерзання теплоносія — за низьких значень температури нагрівач активується незалежно від обраного режиму. Вимкнення здійснюється при досягненні верхнього порогу температури з урахуванням гістерезису, що забезпечує енергоефективну та безпечну роботу вузла.

Об’єднання цих алгоритмів дозволяє системі підтримувати необхідний тепловий комфорт у басейні та компенсувати недостатнє сонячне нагрівання.

3.4.4. Алгоритм запобігання перегріву теплоносія. У випадках перевищення критичної температури на виході теплообмінника система переходить у режим скидання надлишкового тепла в аварійний бак. Це передбачає відкриття клапана, запуск циркуляції та примусове вимкнення електричного нагрівача. Після зниження температури до безпечного рівня клапан повертається у закритий стан. Такий алгоритм захищає гідравлічний контур від перегріву та зберігає працездатність обладнання.

3.4.5. Узагальнений алгоритм регулювання. Комплекс алгоритмів працює як багатоступенева система. На першому етапі здійснюється приймання та перевірка даних. Далі відбувається аналіз теплових умов, вибраних режимів та обмежень. Після цього формуються керуючі впливи на циркуляційний насос, насос басейну, електричний нагрівач та запобіжний клапан, з урахуванням часових затримок, гістерезису та захисних механізмів. Узгоджена взаємодія цих алгоритмів забезпечує ефективну, стабільну та безпечну роботу системи в умовах змінної сонячної активності та різних режимів експлуатації.

3.4.6. Тестування та налагодження програмного забезпечення. Для перевірки коректності роботи системи був створений спеціальний демонстраційний режим, який дозволяє відтворювати типові теплові ситуації без підключення реальних датчиків. У цьому режимі в блоці на базі STM32F103 генерується випадкова поведінка температур що моделює справжню геліосистему. Це значно спростило налагодження логіки, виявлення помилок та перевірку поведінки алгоритмів у межових станах.

Демонстраційний режим (рис. 3.6) дозволив перевірити такі основні сценарії:

- Досягнення порогу вмикання насоса колектора. Коли згенерована різниця температур перевищувала встановлений поріг ΔT_on, система повинна була активувати насос. На дисплеї з’являлося повідомлення про зміну стану.
- Вимкнення насоса після зниження температури. При штучному зменшенні ΔT система мала вимкнути насос лише після досягнення порога гістерезису, що підтверджувало відсутність коливань та антициклювання.
- Увімкнення насоса басейну при низькій температурі води. Симуляція падіння температури в басейні нижче Tmin призводила до активації циркуляційного насоса басейну, що свідчило про правильність роботи режиму підтримання комфортної температури.

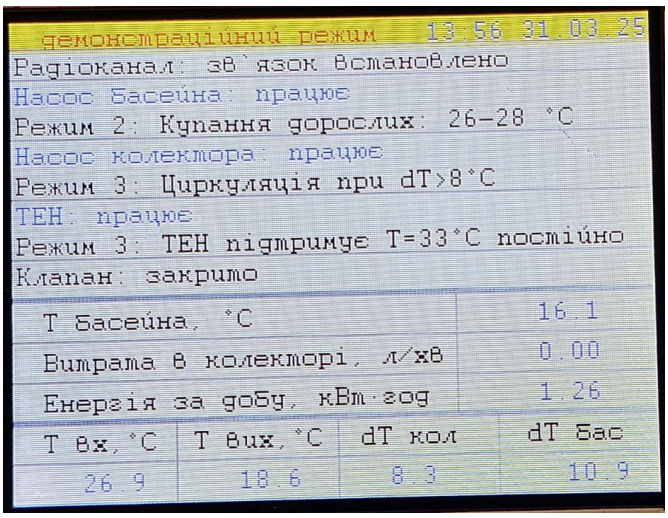

Рис. 3.6. Демонстраційний режим: тестування увімкнення пристроїв

- Поведінка електричного ТЕН. При моделюванні значення температури, нижчого за поріг увімкнення нагрівача, система запускала ТЕН у рамках заданих часових обмежень. Підвищення температури вище Tmax приводило до його вимкнення.

- Аварійні стани. Окремо тестувалися ситуації перегріву теплоносія. При перевищенні критичного значення T_max система переходила у режим захисту: вимикала ТЕН, запускала насос і відкривала клапан скидання тепла. Після повернення температури до безпечного рівня демо-режим підтверджував автоматичне закриття клапана.

Для зручності тестування всі зміни станів відображались на дисплеї, а також дублювалися у вигляді подій з таймштампами у Flash-пам’яті, що дозволяло аналізувати роботу системи після перезавантаження контролера, зображено на рисунку 3.7.

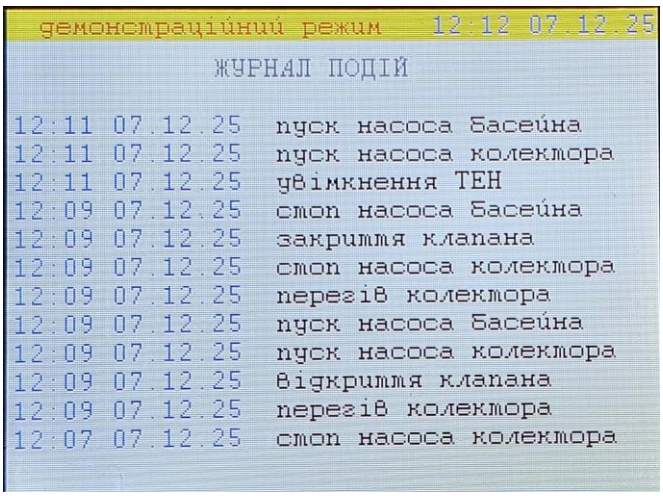

Рис. 3.7. Журнал подій

Налагодження показало, що всі ключові алгоритми — керування насосами, робота ТЕНу, захисні режими та первинна обробка даних — працюють стабільно та відповідають заданим вимогам. Демонстраційний режим став важливим інструментом перевірки логіки, оскільки дозволив визначити критичні моменти роботи без ризику для обладнання та без необхідності створення небезпечних температурних умов у реальній установці.

3.5. Приклад реалізації інтерфейсу моніторингу температури

Приклад роботи графічного інтерфейсу, реалізованого на дисплеї контролера першого мікроконтролера STM32F411CEU6. Інтерфейс забезпечує відображення основних параметрів системи в режимі реального часу та дає користувачу можливість оперативно оцінювати стан сонячного колектора, теплообмінника та басейну.

Завдяки використанню ILI9341 та функцій (printInfoStatic0(), printInfoUpdate0(), printInfoStatic1()) на дисплеї формується багатосторінкове меню з різними інформаційними режимами. На рисунку 3.8 представлено головний інформаційний екран системи. Він містить ключові відомості про стан зв’язку, активні режими роботи, уставки температури та поточні значення

вимірювань. Верхня частина екрана відображає службову інформацію — час, дату та стан радіоканалу. Нижче подається індикація роботи насосів, ТЕН.

Нижня частина екрана містить основні вимірювальні параметри:

- температуру води в басейні;
- витрату теплоносія у колекторному контурі;
- накопичену енергію за добу;
- температуру на вході та виході теплообмінника;
- розраховану різницю температур (dT) для колектора та басейну.

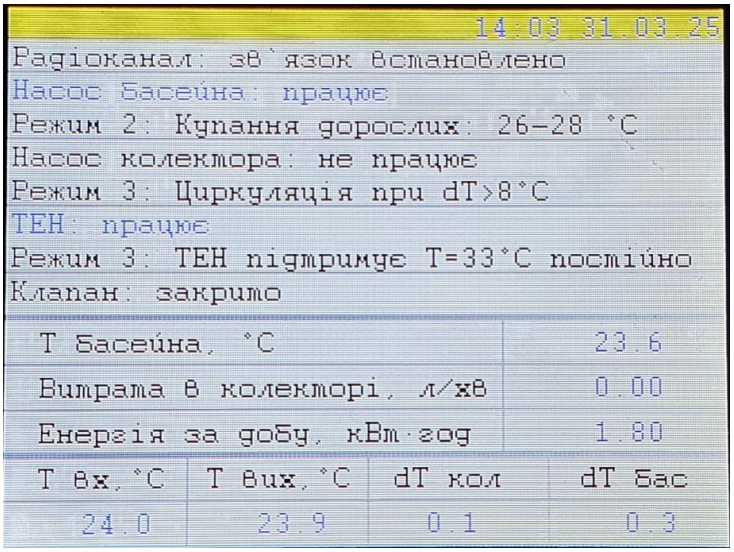

Рис. 3.8. Головний інформаційний екран системи

На другому інформаційному екрані (рис. 3.13) відображаються налаштовувані параметри системи та діагностичні відомості, призначені для контролю температурних уставок, аналізу граничних значень і перевірки коректності обміну даними між модулями.

У верхній частині екрана наведено основні робочі параметри, зокрема значення гістерезису температури, мінімальну різницю температур для запуску циркуляції в контурі колектора, поріг увімкнення циркуляції басейну та максимально допустиму температуру теплообмінника. Зазначені параметри використовуються алгоритмами регулювання та можуть змінюватися користувачем через меню налаштувань.

Середня частина екрана містить інформацію про зафіксовані максимальні та мінімальні значення температур у басейні та на вході й виході теплообмінника із зазначенням дати та часу вимірювання. Це дозволяє оцінити теплові процеси в системі та виявити можливі відхилення в роботі.

У нижній частині екрана відображаються службові дані, зокрема час останнього збереження інформації та кількість помилок радіообміну під час передачі й приймання даних. Ці відомості використовуються для діагностики стабільності зв’язку та коректності збереження даних у пам’яті контролера.

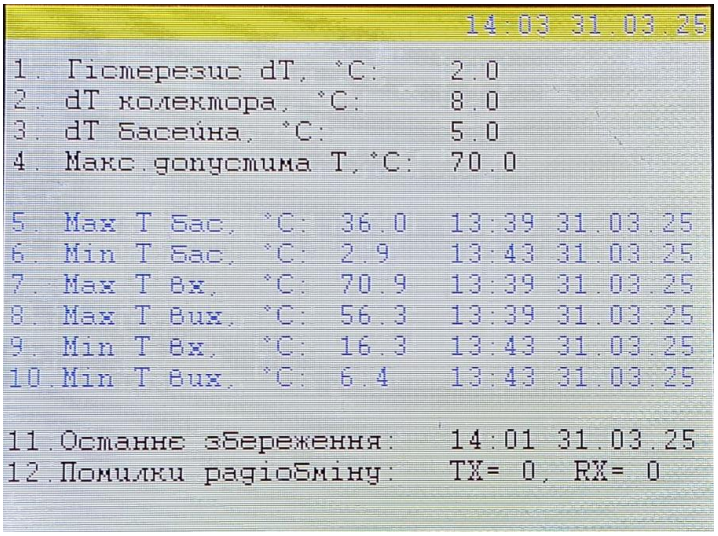

Рис. 3.9. Екран параметрів та діагностики

На рисунку 3.14 наведено екран перегляду експлуатаційної статистики та технічного стану мікроконтролера, який використовується для контролю тривалості роботи виконавчих вузлів і оцінки навантаження на обладнання.

У верхній частині екрана відображається службова інформація про стан мікроконтролера, зокрема його поточна температура та напруга живлення, що дозволяє оцінити тепловий режим і стабільність джерела живлення під час роботи системи.

Середня частина екрана містить накопичену статистику за весь період експлуатації, включаючи сумарну тривалість роботи системи, час роботи насоса

басейна, насоса колектора та електричного ТЕН. Ці дані використовуються для оцінки навантаження на обладнання та планування технічного обслуговування.

У нижній частині екрана наведено інформацію про останню сесію роботи та періоди простою виконавчих механізмів, що дозволяє аналізувати фактичну поведінку системи та виявляти можливі відхилення в її роботі.

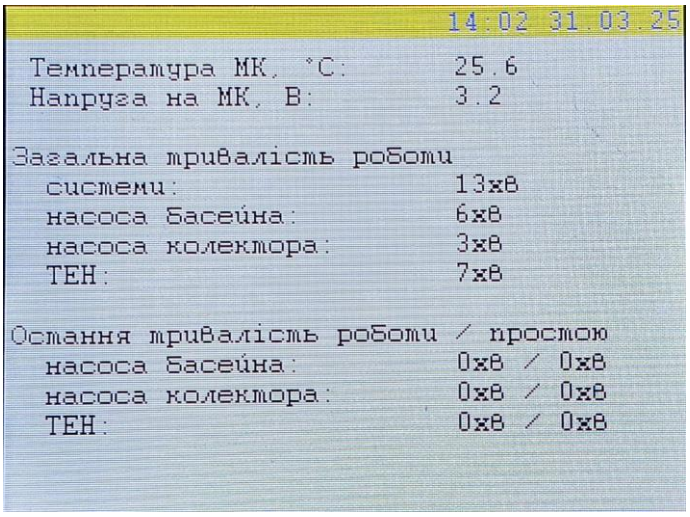

Рис. 3.10. Екран статистики роботи системи

На рисунку 3.11 представлено графічний екран, призначений для аналізу добових змін основних теплотехнічних параметрів системи. На відміну від текстових екранів, що відображають лише поточні значення, цей режим дає можливість оцінити динаміку роботи сонячного колектора й теплообмінного контуру, виявити тенденції, пікові навантаження та періоди спаду температури чи витрати теплоносія.

Екран поділено на чотири інформаційні області:

1. Графік температури (верхня ліва частина).

Тут відображаються зміни температури у визначених точках системи протягом останнього відрізка часу. Користувач може простежити, як змінювалася температура колектора, виходу теплообмінника або басейну залежно від інсоляції, роботи насосів чи ввімкнення нагрівача. Діапазон позначено інтервалом від +0 °C до +50 °C.

2. Графік добової енергії (верхня права частина).

Цей графік показує теплову енергію, що була отримана системою. Значення подаються у джоулях, а вертикальна шкала дозволяє оцінити загальний добовий приріст енергії. Такий показник є корисним для визначення ефективності роботи сонячного колектора.

3. Графік різниці температур ΔT (нижня ліва частина).

Цей графік демонструє зміну температурної різниці між входом та виходом теплообмінника, та між баком та басейном що є ключовими параметроми для запуску або зупинки циркуляції. Діапазон представлений у межах від –5 °C до +15 °C, що дозволяє оцінити ефективність нагрівання та стабільність тепловіддачі.

4. Графік витрати теплоносія (нижня права частина).

Відображає зміну витрати у літрах за хвилину. Шкала до 100 л/хв дозволяє простежити, як поведінка насоса впливає на циркуляцію. Цей показник корисний при аналізі підвищених навантажень або зниження продуктивності контуру.

Поєднання цих чотирьох графічних областей забезпечує комплексне уявлення про роботу системи та дозволяє швидко виявити аномалії, пікові режими чи нехарактерні коливання параметрів.

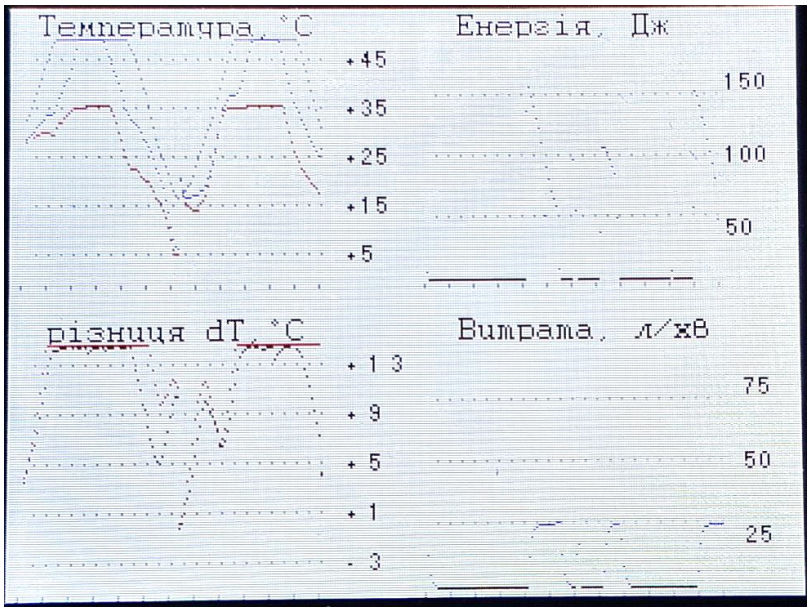

Рис. 3.11. Екран графічного відображення параметрів

3.6. Висновки до розділу 3

У розділі розроблено та реалізовано програмне забезпечення системи автоматизованого контролю й регулювання роботи сонячного колектора. На основі вимог і особливостей апаратної частини сформовано програмну архітектуру, що включає модулі зчитування вимірювальних даних, алгоритми прийняття рішень, механізми взаємодії між контролерами та інтерфейс користувача.

Реалізовано зчитування температурних та експлуатаційних параметрів із польового модуля через радіоканал NRF24L01, а також їх коректну обробку з перевіркою достовірності й обробкою аварійних ситуацій. Алгоритм регулювання температури забезпечує автоматичне керування насосами, електричним нагрівачем і запобіжним клапаном з урахуванням різниць температур, порогових значень і часових обмежень.

Розроблено механізм збереження статистичних даних у зовнішній SPI Flash-пам’яті та графічний інтерфейс для відображення поточних параметрів, станів виконавчих механізмів і налаштувань системи. Проведене тестування підтвердило стабільність роботи програмних модулів, надійність обміну даними між контролерами та коректну роботу системи у штатних і аварійних режимах.

Таким чином, програмна частина забезпечує необхідний рівень автоматизації, моніторингу та керування і надійну взаємодію з апаратною частиною системи сонячного колектора.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

4.1.1. Загальні вимоги охорони праці при монтажі та експлуатації системи.

Монтаж та експлуатація автоматизованої системи контролю сонячного колектора виконуються відповідно до Закону України «Про охорону праці» [22] та інших чинних нормативних документів. Оскільки система включає електронні модулі, насосне обладнання та трубопроводи з теплоносієм, роботи належать до категорії підвищеної небезпеки.

Перед проведенням монтажних робіт працівники проходять інструктажі, перевірку знань з охорони праці та з електробезпеки згідно з НПАОП 40.1-1.21-98 [23]. Працівник повинен бути ознайомлений із планом виконання робіт, схемами живлення, місцями підключення насосів та елементів автоматики.

Під час робіт необхідно використовувати засоби індивідуального захисту такі як: діелектричні рукавиці, захисні окуляри при монтажі колектора, взуття з нековзною підошвою, захисну каску при роботі на висоті.

Особливу увагу приділяють роботам на даху, де розміщуються сонячний колектор і датчики. Встановлення обладнання виконується у суху погоду, із застосуванням страхувальних систем, відповідно до вимог ДСТУ EN 363:2017 (засоби індивідуального захисту від падіння з висоти).

Експлуатація системи допускається після огляду та перевірки:

- цілісності кабельної ізоляції;
- відсутності протікання у трубопроводах;
- надійності кріплення датчиків на колекторі та в системі водопостачання;
- роботоздатності блоків живлення та автоматики.

4.1.2. Вимоги безпеки при роботі з електрообладнанням. Система містить низьковольтні ланцюги керування (5–12 В) та силові кола для насоса і ТЕНа (220 В). Тому необхідно дотримуватися вимог НПАОП 40.1-1.32-01 «Правила охорони праці під час експлуатації електроустановок споживачів» [24].

Основні технічні вимоги до безпечної експлуатації електрообладнання охоплюють комплекс норм і правил, серед них:

- застосування автоматичних вимикачів та пристрою захисного вимкнення, відповідно до вимог ПУЕ-2017 [25];
- гальванічне розділення між мікроконтролером та силовими реле;
- захисне заземлення всіх металевих частин обладнання згідно з ДСТУ EN 60364-5-54:2014 [26];
- використання кабелів, що мають відповідний клас захисту за ДСТУ HD 60364;
- розміщення силових елементів у закритих вологозахищених корпусах не нижче IP54.

Організаційні вимоги охоплюють комплекс заходів, спрямованих на забезпечення безпечної організації робочого процесу персоналу. До таких вимог належать:

- працювати з увімкненими електроустановками можуть лише особи з групою допуску не нижче II;
- усі роботи з ремонту виконуються після знеструмлення всієї системи;
- проводиться періодична перевірка опору ізоляції та надійності заземлювальних провідників;
- заборонено експлуатацію обладнання в умовах конденсації або потрапляння води на корпуси.

Дотримання цих вимог дозволяє знизити ймовірність ураження електричним струмом та забезпечити стабільну роботу системи.

4.1.3. Заходи щодо запобігання ураженню електричним струмом. Захист від ураження електричним струмом є одним із ключових аспектів забезпечення безпечної роботи системи контролю сонячного колектора, оскільки обладнання функціонує за умов підвищеної вологості, впливу атмосферних факторів та використання як низьковольтних, так і силових електричних кіл. Заходи поділяються на технічні та організаційні.

Технічні заходи захисту:

- застосування пристрою захисного вимкнення та автоматичних вимикачів для відключення пошкоджених ділянок [25];
- використання подвійної ізоляції згідно з ДСТУ IEC 61140:2010 [27];
- оптична розв’язка в керуючих ланцюгах і використання силових реле з надійною комутаційною здатністю;
- установка термозахисту ТЕНа, аварійного клапана та контролю перегріву;
- автоматичне вимкнення насоса при відсутності протоку.

Повинні здійснюватися організаційні заходи захисту:

- ведення журналу оглядів обладнання;
- планові випробування електроустановок;
- заборона самостійного втручання користувачів у систему керування;
- регулярне навчання персоналу правилам електробезпеки.

4.1.4. Екологічні аспекти використання сонячних колекторів. Використання сонячних колекторів відповідає основним положенням Національної енергетичної стратегії України до 2050 року [28], згідно з якою розвиток відновлюваних джерел енергії є одним з ключових напрямів зменшення антропогенного впливу на довкілля.

Основні екологічні переваги:

- зменшення споживання електричної енергії та природного газу;
- скорочення викидів CO_2 та інших продуктів згоряння у довкілля;
- зниження навантаження на міські енергетичні мережі;

- тривалий термін служби колекторів та низькі експлуатаційні витрати;
- мінімальна кількість відходів, відсутність забруднюючих речовин у процесі роботи.

Утилізація елементів системи здійснюється відповідно до Закону України «Про управління відходами» [29] та до екологічних вимог щодо поводження з електронними відходами, зокрема роздільного збору металу, скла та полімерів, що відповідає нормативам техноекологічної безпеки [33].

4.2. Безпека в надзвичайних ситуаціях

4.2.1. Запобігання негативному впливу стихійних лих та промислових аварій. Основні параметри їх уражаючої дії на людей, тварин, рослин. Безпека в надзвичайних ситуаціях (НС) є одним із ключових аспектів експлуатації електротехнічного обладнання та систем автоматизованого контролю, зокрема таких, як система управління сонячним колектором. Надзвичайні ситуації можуть виникати внаслідок природних, техногенних, соціальних або комбінованих факторів [22]. Тому система повинна бути не лише надійною у штатному режимі, але й здатною забезпечити захист користувачів та обладнання за умов аварійних, екстремальних та непередбачуваних подій.

До можливих НС, що можуть вплинути на роботу системи, належать:

- пожежі внаслідок короткого замикання або перегріву обладнання;
- електротравматизм через порушення ізоляції чи відсутність заземлення [23];
- гідравлічні аварії — прорив трубопроводів, відмова насоса, надлишковий тиск;
- перегрів теплоносія (вище 70–90 °C), що може призвести до пошкоджень колектора;
- відмова температурних датчиків;
- відсутність електроживлення;

- вплив природних чинників — гроза, удар блискавки, підтоплення, штормовий вітер [22].

Аналіз уражаючих факторів надзвичайних ситуацій дозволяє оцінити рівень ризику та сформувати ефективні заходи щодо їх попередження та мінімізації наслідків [34].

4.2.2. Захист від впливу стихійних лих та техногенних аварій. Апаратні системи захисту дозволяють автоматично вимикати систему при перевищенні струму; контролювати температури; скидати перегріту воду; захищати від зворотної циркуляції для запобігання охолодженню басейну.

Програмні засоби захисту забезпечують аналіз температурних показників; виявлення критичних значень; самодіагностику датчиків; аварійне вимкнення насосів і ТЕН; блокування повторного запуску після аварії; логування подій у Flash-пам’ять [24]. Усі аварійні події дублюються на TFT-дисплей та можуть передаватися по радіоканалу.

Правильні дії користувача є критичними для мінімізації ризиків.

У випадку пожежі потрібно негайно вимкнути електроживлення; застосувати порошковий або вуглекислотний вогнегасник [30]; повідомити службу порятунку за номером 101; евакуюватися згідно з планом евакуації.

У ситуації затоплення або прориву труб потрібно перекрити подачу води; вимкнути насоси та живлення системи; викликати спеціалістів.

У разі ураження електрострумом необхідно відключити живлення; забезпечити безпечний контакт при наданні допомоги; викликати медичну допомогу за номером 103; надати домедичну допомогу відповідно до ДСТУ [26].

Під час природних надзвичайних ситуацій таких як, гроза і блискавка, до системи захисту висуваються підвищені вимоги: наявність заземлення відповідно до ПУЕ [32]; встановлення системи блискавкозахисту [31]; застосування грозозахисних реле.

За умови штормового вітру достатня механічна фіксація панелей, використання анкерних болтів для монтажу на даху та систематична перевірка кріплень конструкцій.

За наявності підтоплень потрібно підняття обладнання над рівнем підлоги; використання герметичних кабельних коробок; контроль вологості приміщення.

При екстремальних температурах (замерзання води під час морозів чи теплове розширення матеріалів в спеку) необхідно використовувати протиморозну рідину у колекторному контурі; передбачити наявність режиму «антизамерзання» в мікроконтролері; провести термоізоляцію труб та баків; забезпечити відведення перегрітої води у аварійний контур.

4.3. Висновки до розділу 4

У розділі проведено аналіз вимог охорони праці та безпеки під час монтажу і експлуатації системи контролю сонячного колектора. Сформовано комплекс технічних і організаційних заходів, що забезпечують мінімізацію ризиків під час роботи з електричним та гідравлічним обладнанням. Визначено екологічні переваги використання сонячних колекторів і описано дії системи у надзвичайних ситуаціях.

Застосування розроблених заходів забезпечує надійну, безпечну та екологічно орієнтовану експлуатацію системи відповідно до чинних нормативів України.

ВИСНОВКИ

Відповідно до поставленого завдання у кваліфікаційній роботі було вирішено науково-практичне завдання, що полягало у розробці методів та програмно-апаратних засобів контролю сонячного колектора для автоматизованого регулювання температури води в басейні з використанням мікроконтролерних технологій.

У результаті досягнення поставленої мети було отримано такі основні результати:

1. Виконано аналіз сучасних методів і засобів контролю температури в системах сонячного нагріву води, а також визначено основні вимоги до автоматизованих систем керування сонячними колекторами для підігріву басейнів.

2. Розроблено узагальнену структуру комп’ютеризованої системи контролю та управління сонячним колектором і виконано обґрунтування вибору елементної бази, зокрема мікроконтролерів STM32, датчиків температури, виконавчих механізмів та засобів відображення інформації.

3. Розроблено електричну принципову схему системи контролю сонячного колектора, яка забезпечує надійне зчитування температурних параметрів, керування насосним обладнанням і реалізацію захисних режимів роботи.

4. Описано алгоритм функціонування системи та розроблено програмне забезпечення для мікроконтролерів, яке реалізує збір і обробку даних від датчиків, автоматичне регулювання температури води в басейні, керування виконавчими пристроями та візуалізацію параметрів у режимі реального часу.

Розроблена комп’ютеризована система контролю сонячного колектора забезпечує автоматичне підтримання заданої температури води в басейні, підвищує ефективність використання сонячної енергії та знижує потребу у додаткових енергетичних ресурсах.

Результати тестування підтвердили, що всі режими роботи системи функціонують у повній відповідності до вимог технічного завдання, а алгоритми керування забезпечують стабільну та безпечну роботу обладнання.

Спроєктована система є енергоефективною порівняно з класичними рішеннями ручного або часово-залежного керування, оскільки дозволяє мінімізувати втрати тепла та зменшити споживання електроенергії насосним обладнанням.

Розроблена комп’ютеризована система може бути використана у приватних басейнах, спортивно-оздоровчих комплексах, готельних господарствах, а також як навчальний і демонстраційний зразок у закладах вищої освіти технічного профілю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль: ТНТУ. 2024. 45 с.

2. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. 2024. 44 с.

3. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 32 с.

4. Невмержицький В. В., Система моніторингу температури води в басейні XIII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» Тернопіль: ТНТУ, 2025. 134 с.

5. Невмержицький В. В., Методи та засоби контролю сонячним водонагрівачем на базі мікроконтролера stm32 XIV міжнародній науково-технічній конференції молодих учених та студентів «Актуальні задачі сучасних технологій» Тернопіль: ТНТУ, 2025. 313 с.

6. Що таке сонячний колектор. Види і принцип роботи. URL: https://xn--e1aamjjfht.com.ua/sonjachni-kolektory-pryncyp-roboty. (17.11.2025)

7. Сонячний колектор: переваги та недоліки сонячної енергії URL: https://teplosfera.com/blog/sonacni-stancii/sonacnij-kolektor-perevagi-ta-nedoliki-sonacnoi-energii. (14.11.2025)

8. Нагрівання басейну сонячним колектором URL: https://reneco.com.ua/stati-ua-solnechnye_kollektory/nagrivannya-baseynu-sonyachnim-kolektorom/. (25.11.2025)

9. ДСТУ EN 12975-1-2001 Системи теплові сонячні та їхні компоненти. Колектори сонячні. Частина 1. Загальні технічні вимоги (EN 12975-1:2000, IDT) URL:https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=66643&utm_source=chatgpt.com. (30.11.2025)

10. Сонячні нагрівачі води (СНВ) сонячний колектор (СК) для СНВ URL: https://elim-ua.com.ua/files/Elim_instr_SK.pdf?utm_source=chatgpt.com. (6.12.2025)

11. Мельничук О. В., Кальниш В. А. Автоматизація процесів теплопостачання з використанням PID-регулювання. К.: КПІ ім. І. Сікорського, 2021. 51 с.

12. Теличко О. С. Мікропроцесорні системи автоматичного регулювання температури. Львів: НУ "Львівська політехніка", 2019. 40 с.

13. Ковальчук С. В. Сенсорні елементи систем автоматизації: навч. посібник. Київ: КПІ ім. І. Сікорського, 2020. 37 с.

14. Лісняк О. С. Мікроконтролерні системи контролю та регулювання температури. Львів: НУ «Львівська політехніка», 2019. 54 с.

15. Куценко О. П. Апаратне забезпечення систем збору даних у сонячній енергетиці. Харків: ХНУРЕ, 2020. 44 с.

16. Дорошенко А. В., Нікітін В. В. Теплотехнічні системи з сонячними колекторами: аналіз та методи керування. Харків: ХНУРЕ, 2020. 31 с.

17. Григоренко В. М. Arduino в системах автоматизації: навч. посібник. Київ: КПІ ім. І. Сікорського, 2021. 56 с.

18. Теличко О. С. Мікроконтролерні системи автоматизації теплотехнічних установок. Львів: НУ «Львівська політехніка», 2020. 57 с.

19. Мельничук О. В., Кальниш В. А. Автоматизація процесів теплопостачання з використанням мікроконтролерів Arduino. Київ: КПІ ім. І. Сікорського, 2021. 27 с.

20. Сонячні електростанції від виробника URL: https://www.solarenergyua.com/ (12.11.2025)

21. Мельничук О. В., Кальниш В. А. Автоматизація процесів теплопостачання з використанням мікроконтролерів Arduino. Київ: КПІ ім. І. Сікорського, 2021. 26 с.

22. Кодекс цивільного захисту України, 2012. (22.11.2025)

23. НПАОП 40.1-1.21-98. Правила безпечної експлуатації електроустановок. (13.11.2025)

24. НПАОП 40.1-1.32-01. Правила охорони праці під час експлуатації електроустановок споживачів. (15.11.2025)

25. Правила улаштування електроустановок (ПУЕ), 2017. (17.11.2025)

26. ДСТУ EN 60364-5-54:2014. Електроустановки будівель. (19.11.2025)

27. ДСТУ IEC 61140:2010. Захист від ураження електричним струмом. (19.11.2025)

28. Енергетична стратегія України до 2050 року. (23.11.2025)

29. Закон України «Про управління відходами». (24.11.2025)

30. ДСТУ EN 3-7:2015. Вогнегасники. (22.11.2025)

31. ДСТУ EN 62305-3:2012. Блискавкозахист. (18.11.2025)

32. ПУЕ — Правила улаштування електроустановок, чинна редакція (6.11.2025)

33. Стручок В.С. Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник. Тернопіль: ТНТУ ім. І.Пулюя, 2022. 40 с.

34. Стручок В.С. Безпека в надзвичайних ситуаціях. Методичний посібник. Тернопіль: ФОП Паляниця В.А., 2022. 16 с.

35. Тиш Є.В., Гончаренко О.Р. Алгоритм автоматизованого режиму роботи сонячного трекера. International Scientific Journal Grail Of Science. №10, Vinnytsia-Vienna, 2021. P.268-271

36. Гончаренко О.Р, Тиш Є.В. Системи керування сонячних трекерів. Матеріали X міжнародної науково-практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (24-25 листопада

2021р.).Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2021. с. 90.

37. Стадник Н. Б., Осухівська Г. М. Мікроконтролерні засоби керування енергоефективними теплотехнічними системами Актуальні задачі сучасних технологій : матеріали XIV міжнар. наук.-техн. конф. молодих учених та студентів. Тернопіль : ТНТУ ім. І. Пулюя, 2024. 112 с.

38. Осухівська Г. М. Інформаційно-вимірювальні системи в енергетиці : навч. посіб. Тернопіль : ТНТУ ім. І. Пулюя, 2021. 148 с.

39. Климчук О. В., Стадник Н. Б. Аналіз ефективності використання сонячних колекторів у системах гарячого водопостачання Вісник ТНТУ. Тернопіль, 2020. 41 с.

ДОДАТОК А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 681.5:004.
В. Невмержицький; Н. Стадник, к. т. н.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

СИСТЕМА МОНІТОРИНГУ ТЕМПЕРАТУРИ ВОДИ В БАСЕЙНІ

UDC 681.5:004.
V. Nevmerzhytskyi; N. Stadnyk, PhD.

POOL WATER TEMPERATURE MONITORING SYSTEM

Система контролю, розроблена для роботи із сонячним колектором, базується на інтеграції двох мікроконтролерів STM32F411CEU6 та STM32F103C8T6, що формують розподілену архітектуру вимірювання та керування. Така побудова забезпечує підвищену надійність, автономність та гнучкість у реалізації алгоритмів теплового регулювання. Температурні параметри знімаються цифровими датчиками DS18B20, розміщеними в ключових точках теплотехнічної системи: на вході та виході сонячного колектора, у баку-накопичувачі та безпосередньо в басейні, що забезпечує повну інформативність про динаміку процесів нагріву [1].

Зібрані дані передаються на основний контролер за допомогою бездротового модуля NRF24L01, що дозволяє мінімізувати кількість дротових з'єднань і підвищити електробезпеку системи. Центральний модуль виконує аналіз температурних градієнтів, розрахунок ефективності нагріву та визначення оптимального режиму циркуляції теплоносія. Контролер реалізує алгоритм диференційного керування ΔT із підтримкою гістерезису, що дозволяє зменшити кількість непотрібних запусків насосів, запобігти охолодженню бака та забезпечити стабільний температурний режим води басейну.

Для підвищення надійності в систему інтегровані алгоритми валідації даних, автоматичне виявлення відмов датчиків, а також захисний контур, який активується при перегріві теплоносія. Візуалізація показників здійснюється на TFT-дисплеї ILI9341 у вигляді багатосторінкового інтерфейсу, що надає користувачу повну інформацію про стан системи в реальному часі [2].

Експериментальні дослідження підтвердили, що використання автоматизованої мікроконтролерної системи дозволяє підвищити енергоефективність роботи сонячного колектора та зменшити кількість циклів увімкнення насосів на 25–30%. Розроблене рішення характеризується масштабованістю, що дає можливість його модернізації шляхом додавання нових сенсорів, модулів зв'язку або інтеграції в системи «розумного дому».

Література

1. Duffie J. A., Beckman W. A. Solar Engineering of Thermal Processes. Wiley, 2013. URL: https://www.wiley.com/en-us/Solar+Engineering+of+Thermal+Processes (дата звернення: 08.12.2025).
2. ILI9341. Технічний опис драйвера TFT-дисплея. Ilitek Corporation, 2011. URL: https://www.displayfuture.com/Display/datasheet/controller/ILI9341.pdf (дата звернення: 08.12.2025).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедиміната (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

2. Третяк В. Ф., Пашнєва А. А., Оптимізація структури сховища даних у вузлах інфокомунікаційної мережі хмарного середовища, 2017. URL: https://journals.nupp.edu.ua/sunz/article/view/390 (дата звернення: 25.11.2025).

3. Денисенко А.В., Козлов О.В. Аналіз технологій та алгоритмів управління обчислювальними ресурсами при обробці відеоданих. MDCS, 2023. 229 с. URL: https://card-file.ontu.edu.ua/server/api/core/bitstreams/9cde9e98-790b-464c-bda6-13d70d592a01/content (дата звернення: 25.11.2025).

4. Гмиря І. О. Хмарні технології у розробці системи з відео аналізу. НТУ «ХПІ», 2024. URL: https://openarchive.nure.ua/entities/publication/23c56163-62ba-425c-9124-75d879dcfb5f (дата звернення: 25.11.2025).

5. Шматко О.В., Гамаюн І.П., Коломійцев О.В., Третяк В.Ф., Рудаков І.С., Бердочник А.Д. ДОСЛІДЖЕННЯ ТА ОЦІНКА ПІДСИСТЕМИ ВИЯВЛЕННЯ ТА КЛАСИФІКАЦІЇ ОБ'ЄКТІВ У ВІДЕОПОТОЦІ. Системи обробки інформації. 2025. № 4 (179). С. 70-80. URL: https://doi.org/10.30748/soi.2024.179.08.

УДК 004.9:697.14:628.97

В.В. Невмержицький, Н.Б. Стадник, к.т.н.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

МЕТОДИ ТА ЗАСОБИ КОНТРОЛЮ СОНЯЧНИМ ВОДОНАГРІВАЧЕМ НА БАЗІ МІКРОКОНТРОЛЕРА STM32

V.V. Nevmerzhytskyi, N.B. Stadnyk, PhD.

METHODS AND MEANS OF CONTROLLING A SOLAR WATER HEATER BASED ON THE MICROCONTROLLER STM32

Сучасні тенденції переходу до використання відновлюваних джерел енергії створюють потребу у впровадженні автоматизованих систем керування технологічними процесами, зокрема систем нагріву води на основі сонячних колекторів. Ефективність роботи таких установок значною мірою залежить від точності вимірювання температурних параметрів, коректної оцінки теплових потоків та раціонального керування циркуляцією теплоносія. Це особливо актуально для системи підігріву води в басейні, де необхідно забезпечувати стабільний і безпечний температурний режим за мінімальних витрат енергії [1].

Розроблена система контролю сонячного колектора базується на мікроконтролері STM32, який здійснює зчитування, обробку та аналіз даних від цифрових датчиків температури DS18B20, розташованих у ключових точках: на вході та виході колектора у баку-накопичувачі та в басейні. Використання інтерфейсу OneWire дозволяє підключати кілька датчиків на одній шині при збереженні високої точності вимірювань (±0,5 °C), що спрощує побудову та масштабування системи.

Алгоритм керування грунтується на аналізі різниць температур між входом і виходом колектора і між басейном та баком. У разі, коли температура в баці перевищує температуру у басейні на встановлену користувачем величину, тоді мікроконтролер активує реле для вмикання насоса басейна. Коли різниця температур зменшується до нижнього порогу, насос басейна вимикається, що запобігає небажаному охолодженню бака та підвищує енергоефективність. Реалізовано також механізм запобігання частому перемиканню насосів, що забезпечує плавну й стабільну роботу системи навіть за швидких змін температури. Система містить режими самодіагностики, захисту від перегріву та виявлення відмови датчиків.

Візуалізація параметрів здійснюється на TFT-дисплеї з драйвером ILI9341, який відображає температуру в контрольних точках, стан насосів, величину ΔT та повідомлення про помилки. Передбачено можливість передавання даних на віддалений пристрій за допомогою радіомодуля на частоті 2,4 ГГц, що забезпечує дистанційний моніторинг та дистанційне керування.

Математична модель теплообміну, реалізована у програмному забезпеченні, ґрунтується на безперервному аналізі температур на вході (Твх) і виході (Твих) колектора та температури води в басейні. На основі цих параметрів система визначає реальний тепловий потенціал колектора і приймає рішення щодо доцільності увімкнення насоса колектора. Якщо колектор не забезпечує достатнього нагріву (Твих-Твх < ΔT), насос не активується, що запобігає зайвому енергоспоживанню і дозволяє системі працювати у режимі природної циркуляції. Такий підхід зменшує енергоспоживання, знижує зношення обладнання та підвищує загальну ефективність геліосистеми.

Література

1. Duffie J. A., Beckman W. A. Solar Engineering of Thermal Processes. Wiley, 2013. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118671603 (дата звернення: 27.11.2025).

УДК 069:004

В.В. Никитюк, к.т.н., доц.; І. К. Генсьора
(Тернопільський національний технічний університет імені Івана Пулюя)

ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ ТА ОРГАНІЗАЦІЯ ДАНИХ У СИСТЕМАХ ЦИФРОВОГО ЗБЕРЕЖЕННЯ МУЗЕЙНИХ ФОНДІВ

V.V. Nykytyuk, Ph.D, Assoc. Prof; I. K. Hensora

INFORMATION MODELING AND DATA ORGANIZATION IN DIGITAL PRESERVATION SYSTEMS FOR MUSEUM COLLECTIONS

Цифрове збереження культурної спадщини сьогодні є одним із ключових напрямів розвитку музейної сфери. Воно передбачає створення систем, у яких інформація про музейні об'єкти структурується, зберігається та стає доступною через відкриті цифрові середовища. Однією з найвідоміших є CIDOC Conceptual Reference Model (CIDOC CRM) онтологічна модель, що дозволяє уніфікувати опис об'єктів і забезпечує семантичну сумісність між музеями, архівами та бібліотеками [1]. Завдяки цій моделі культурні інституції можуть обмінюватися даними незалежно від програмних платформ чи мов опису.

Сучасні інформаційні системи дедалі частіше базуються на технологіях семантичного вебу та онтологічного моделювання, які формують новий рівень представлення знань. Такий підхід реалізовано, зокрема, у проєктах Europeana та Google Arts & Culture, що забезпечують єдиний доступ до мільйонів цифрових об'єктів з усього світу [2]. Особливе значення у цифровому збереженні мають метадані структуровані описи цифрових об'єктів, які визначають їх зміст, контекст, технічні характеристики та історію змін. Стандарт PREMIS (Preservation Metadata Implementation Strategies), розроблений Бібліотекою Конгресу США, визначає ключові елементи опису життєвого циклу цифрового об'єкта.[3]. PREMIS активно

ДОДАТОК Б

Код програмного забезпечення

```
/* USER CODE BEGIN Header */
/**********************************************************
 * @file  : main.c
 * @brief : блок 1 - МК STM32F411 - версія від 07.12.2025
  **********************************************************/
/* Includes ------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "rtc.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"

#include "spif.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "NRF24.h"
#include "NRF24_reg_addresses.h"
#include "NRF24_conf.h"
#include "keyboard.h"         //модуль роботи з кнопками
#include "display.h"          //драйвер роботи з дисплеями по SPI
#include "ili9341.h"          //драйвер для дисплея ILI9341
#include "mystring.h"         //бібліотека роботи з рядками
#include <math.h>
#include <stdbool.h>
#include <limits.h>
#include "stm32f4xx_hal_exti.h"
#include "stm32f4xx_hal_cortex.h"

#define USE_ADC

#define DEBUG 1

#define ERROR_SENSOR  INT16_MAX    // значення для передачі
непрацездатності датчика температури
//для передачі коду помилки передаєм значення INT16_MAX -
sensorStatus[MAXDEVICES_ON_THE_BUS]
//uint8_t sensorStatus[MAXDEVICES_ON_THE_BUS];
// 0 = OK
// 1 = not present
// 2 = ROM CRC error
// 3 = scratchpad CRC error
// 4 = invalid temperature (85 / -127)
// 5 = read error

//функція для виводу відладочної інформації
```

```
int _write(int file, uint8_t *ptr, int len)
{
     for (int i = 0; i < len; i++)
     {
          ITM_SendChar(*ptr++);
     }
     return len;
}

// Конфігурація флеш-пам'яті (SPIF)
#define FLASH_SECTOR_SIZE      4096    // 4KB
#define FLASH_TOTAL_SECTORS    2048    // Кількість секторів Для 8MB
#define CHART_DRY_SECTORS      1024    // виділяємо 4MB для
CHART_DRY
#define SETTINGS_SECTORS       128      //виділяємо 512KB для
DataForSave
#define METADATA_SECTOR        (FLASH_TOTAL_SECTORS - 1) // Останній
сектор для метаданих

#define CHART_DRY_START_ADDR   0x000000
#define SETTINGS_START_ADDR    (CHART_DRY_START_ADDR +
(CHART_DRY_SECTORS * FLASH_SECTOR_SIZE))     // = 4 194 304
#define METADATA_START_ADDR    (METADATA_SECTOR * FLASH_SECTOR_SIZE)
     // = 8 384 512
#define METADATA_SIZE          (sizeof(FlashMetadata) * 2) // 8 x 2
= 16 байт
#define METADATA_MAX_POSITIONS (FLASH_SECTOR_SIZE / METADATA_SIZE)
// 256
#define ERROR_POSITION         (UINT16_MAX - 1) // Помилка читання
#define EMPTY_SECTOR           UINT16_MAX      // Чистий сектор

#define CHART_DRY_SIZE          (sizeof(CHART_DRY))
#define DataForSave_SIZE        (sizeof(DataForSave))

//В функції static void MX_RTC_Init(void) є дві структури, їх
оголошуємо як глобальні для виводу часу, дати
RTC_TimeTypeDef sTime;
RTC_DateTypeDef sDate;

//======== NRF24 =============
#define PLD_SIZE 8  //розмір пакета передавання/приймання в байтах
uint8_t tx_addr[5] = {0x11, 0xF0, 0x34, 0x35, 0x54};//адреса NRF24
volatile uint8_t irq_RX = 0; //ознака отримання параметрів з
датчиків
volatile uint8_t irq_3sec = 0;//перевіряємо чи є переривання від 1-
го таймера TIM1 раз в 3 сек
uint8_t Start_Info_time = 2;  // лічильник періодів по 3 сек для
відображення напису на екрані при старті
volatile uint8_t error_RX = 2;     //кількість невдалих передач
даних радіомодулем
volatile uint16_t error_TX = 2;    //кількість невдалих отримань
даних радіомодулем
```

```
//================================
#ifdef  USE_ADC     //для вимірювання температури та напруги МК
#define NUM_SAMPLES 100
// Функція для читання ADC (аналого-цифровий конвертер) для
вимірювання температури та напруги МК
uint32_t Read_ADC(ADC_HandleTypeDef *hadc, uint32_t channel) {
    ADC_ChannelConfTypeDef sConfig = {0};
    sConfig.Channel = channel;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_480CYCLES;
    HAL_ADC_ConfigChannel(hadc, &sConfig);
    HAL_ADC_Start(hadc);
    HAL_ADC_PollForConversion(hadc, 100);
    uint32_t raw = HAL_ADC_GetValue(hadc);
    HAL_ADC_Stop(hadc);
    return raw;
}
#endif

//================================
//  Ознаки завершення ініціалізації , якщо = 1
uint8_t system_ready = 0; //  = 0, якщо ще не завершилась початкова
ініціалізація
uint8_t temp_ready = 0; //  = 0, ще не було отримано даних з
датчиків по радіоканалу

uint8_t dataRX[PLD_SIZE];     //масив для для збереження отриманих
даних
uint8_t dataTX[PLD_SIZE];     //масив для для збереження даних для
відправки

// структура Packet_RX для збереження значень отриманих по
радіоканалу
typedef struct Packet_RX
{
     int16_t        t1;       // поточна температура на вході в
теплообмінник - в сотих градуса
     int16_t        t2;       // поточна температура на виході з
теплообмінника - в сотих градуса
     int16_t        t3;       // поточна температура в басейні - в
сотих градуса
     int16_t        V;        // кількість імпульсів датчика витрати
за останні три секунди
}    Packet_RX;

Packet_RX packet_RX = {
    .t1   = 2300,
    .t2   = 2300,
    .t3   = 2300,
    .V   = 0,
};
```

```
// структура Packet_TX для збереження значень для передавання по
радіоканалу
typedef struct Packet_TX
{
    uint8_t flags;
    uint8_t payload[PLD_SIZE - 1];
} Packet_TX;

Packet_TX packet_TX = {
    .flags   = 0x00,
    .payload = {0}
};

// Структура для збереження даних на flash
typedef struct DataForSave
{
     int8_t    mode_flow;          // режим роботи насоса сонячного
колектора
     int8_t    mode_heat;          //режим роботи ТЕН
     uint32_t hour_unit;      // загальний час роботи - вимірюється
в інтервалах сканування датчиків = TIME_SCAN_SENSOR
     uint32_t hour_motor;     // час роботи насоса сонячного
колектора - вимірюється в інтервалах сканування датчиків =
TIME_SCAN_SENSOR
     uint32_t hour_heat;      // час роботи ТЕН (трубчастий
електричний нагрівач) - вимірюється в інтервалах сканування датчиків
= TIME_SCAN_SENSOR
     int16_t  tOutMin, tInMin, tPoolMin;     // Мінімальні
температури за період спостереження - в сотих градуса
     int16_t  tOutMax, tInMax, tPoolMax;     // Максимальні
температури за період спостереження - в сотих градуса
     uint8_t tOutMinTimeDate[5];   //minute,hour,day,month,year
     uint8_t tOutMaxTimeDate[5];   //minute,hour,day,month,year
     uint8_t tInMinTimeDate[5];    //minute,hour,day,month,year
     uint8_t tInMaxTimeDate[5];    //minute,hour,day,month,year
     uint8_t tPoolMinTimeDate[5];  //minute,hour,day,month,year
     uint8_t tPoolMaxTimeDate[5];  //minute,hour,day,month,year
     uint8_t SaveTimeDate[5]; //Час, Дата останнього збереження
даних на flash

     uint8_t        flags;    // байт ознак
                    // 0 біт - насос увімкнено/вимкнено
                    // 1 біт - ТЕН увімкнено/вимкнено
                    // 2 біт - насос басейна увімкнено/вимкнено
                    // 3 біт - дисплей увімкнено
                    // 4 біт - демо-режим увімкнено
                    // 5-7 -  ПУСТО
     int8_t    color_mode;    // Колірна схема: 0, 1, 2, 3
     uint8_t N_Screen;   // Номер поточного інфо екрана

     uint8_t        dT_min_pump;   // поріг увімкнення насоса
колектора по різниці температур в градусах, тільки позитивні
значення
```

```
    uint8_t        dT_min_pump_pool;   // поріг увімкнення насоса
колектора по різниці температур в градусах, тільки позитивні
значення

    int16_t calibrationRTC;
//   часові інтервали роботи ТЕН
    uint8_t PeriodTime_1[4]; //start hour, start minute, end hour,
end minute
    uint8_t PeriodTime_2[4];
    uint8_t PeriodTime_3[4];

    int8_t Use_Period_1;
    int8_t Use_Period_2;
    int8_t Use_Period_3;

    uint32_t Q_day;     //   отримана енергія за останній день
    int8_t    demo_mode;          // Вимкнено = 0 / Увімкнено = 1
демонстраційний режим
    uint8_t   T_max_valve;   // поріг спрацювання запобіжного
режиму, в градусах, тільки позитивні значення
    uint8_t   dT_OFF;             // Гістерезис температури в
градусах

//   для збереження ресурсу насосів і зменшення циклів запуску-
зупинки
    uint8_t   MIN_ON_TIME ;  // мінімальний час роботи насоса після
запуску у хвилинах: 3 хв
    uint8_t   MIN_OFF_TIME;  // мінімальний час між вмикання насоса
у хвилинах: 2 хв

    int8_t    mode_pump_pool;          // режим роботи насоса
басейна
    uint32_t hour_motor_pool;     // час роботи насоса басейна -
вимірюється в інтервалах сканування датчиків = TIME_SCAN_SENSOR

    uint8_t   T_min_pool[6];      // мінімальне значення
температури в басейні для режимів, в градусах, тільки позитивні
значення
    uint8_t   T_max_pool[6];      // максимальні значення
температури в басейні для режимів, в градусах, тільки позитивні
значення

}    DataForSave;

DataForSave *settingRAM;

DataForSave* DataForSaveNew(void)
{
    DataForSave *stng = (DataForSave*)calloc(1,
sizeof(DataForSave));
    stng->mode_flow          = 1;
    stng->hour_unit          = 0;
    stng->hour_motor    = 0;
```

```
	stng->tOutMin       = 9999;
	stng->tInMin        = 9999;
	stng->tOutMax       = -5555;
	stng->tInMax        = -5555;
	stng->tPoolMin      = 9999;
	stng->tPoolMax      = -5555;
	stng->flags              = 0x00;
	stng->color_mode    = 3;
	stng->mode_heat          = 1;

	stng->N_Screen      = 1;

	stng->dT_min_pump   = 8;
	stng->dT_min_pump_pool   = 5;
	stng->T_max_valve   = 70;

	uint8_t   var;
	for(int i=0;i<5;i++)
	{
		switch (i)
		{
			case  0:                  //minute
			case  1:  var = 0;  break;    //hour
			case  2:                  //day
			case  3:  var = 11; break;    //month
			case  4:  var = 25; break;    //year
		}

		stng->tOutMinTimeDate[i] = var;
		stng->tOutMaxTimeDate[i] = var;
		stng->tInMinTimeDate[i] = var;
		stng->tInMaxTimeDate[i] = var;
		stng->tPoolMinTimeDate[i] = var;
		stng->tPoolMaxTimeDate[i] = var;

		stng->SaveTimeDate[i] = var;
	}

	stng->calibrationRTC = 0;

	//start hour, start minute, end hour, end minute
	memcpy(stng->PeriodTime_1, (uint8_t[]){06,00, 07,00},
sizeof(stng->PeriodTime_1));
	memcpy(stng->PeriodTime_2, (uint8_t[]){12,00, 13,00},
sizeof(stng->PeriodTime_2));
	memcpy(stng->PeriodTime_3, (uint8_t[]){18,00, 22,00},
sizeof(stng->PeriodTime_3));

	stng->Use_Period_1 = 1;
	stng->Use_Period_2 = 0;
	stng->Use_Period_3 = 0;

	stng->Q_day  = 0;
```

```
    stng->demo_mode      = 0;

    stng->dT_OFF = 2;

    stng->MIN_ON_TIME = 3 ;  // мінімальний час роботи насоса після
запуску у хвилинах: 3 хв
    stng->MIN_OFF_TIME = 2;  // мінімальний час між вмикання насоса
у хвилинах: 2 хв

    stng->mode_pump_pool = 6;          // режим роботи насоса
басейна
    stng->hour_motor_pool = 0;    // час роботи насоса басейна -
вимірюється в інтервалах сканування датчиків = TIME_SCAN_SENSOR

// мінімальне значення температури в басейні для режимів, в
градусах, тільки позитивні значення
    memcpy(stng->T_min_pool, (uint8_t[]){24, 26, 28, 30, 23, 19},
sizeof(stng->T_min_pool));

// максимальні значення температури в басейні для режимів, в
градусах, тільки позитивні значення
    memcpy(stng->T_max_pool, (uint8_t[]){26, 28, 30, 32, 33, 20},
sizeof(stng->T_max_pool));

    return stng;
}

int16_t    tick_SPI_Flash;         // Змінна для збереження часу що
пройшло після останнього збереження статистики,
                    //   порівнюємо з TIME_HOUR і обнуляємо змінну
та зберігаємо статистику


// структура для графіків і логу подій
typedef struct CHART_DRY
{
// 1 день = 24 год. х 60 хв. = 1440 хв.
// точку на графіку ставимо кожні 12 хв
// 1400 хв. / 12 хв. = 120 - кількість точок на шкалі часу для 24
год.
// в масивах зберігаються вирахувані значення (координата Y) для
побудови графіків
    uint8_t tInChart[120];        // Температура на вході в
теплообмінник
    uint8_t tOutChart[120];       // Температура на виході з
теплообмінника
    uint8_t dTChart[120];         // різниця температур колектоа
    uint8_t dT_pool_Chart[120];        // різниця температур
басейна

    uint8_t tPoolChart[120]; // Температура в басейні
```

```
    uint8_t V_Chart[120];          // витрата
    uint8_t Q_Chart[120];          // Отримана енергія

    uint8_t posChart;         // позиція в масиві графіків - початок
виводу від 0 до 120-1
    uint8_t TimeChart;        // час до виводу чергової точки на
графік
    int8_t ChartMotor;        // ознака роботи насоса під час
інтервалу графіка, якщо
                          //насос працював (навіть якщо одне
вимірювання), то на графіку фон міняється
    int8_t ChartHeat;         // ознака роботи нагрівача під час
інтервалу графіка, якщо нагрівач
                          //працював (навіть якщо одне вимірювання),
то на графіку фон міняється

//   графік отриманої енергії за останні 280 днів

    uint8_t E_Chart[280];     // Отримана енергія
    uint8_t posChart_E;       // позиція в масиві графіків - початок
виводу від 0 до 280-1
    uint8_t posDayE;

    uint8_t IventTimeDate[12][5]; //minute,hour,day,month,year

    int8_t    ID_ivent[12];  //код події
    uint8_t CountLogs;  //кількість подій

}    CHART_DRY;

int8_t CoolData;

//функція задання початкових значень структури CHART_DRY
CHART_DRY* CHART_DRY_New(void)
{
    CHART_DRY *chdr = (CHART_DRY*)calloc(1, sizeof(CHART_DRY));
    chdr->posChart = 0;
    chdr->TimeChart     = 0;
    chdr->ChartMotor= 0;
    chdr->ChartHeat = 0;
    chdr->posChart_E = 0;
    chdr->posDayE = 0;

    //початкові значення для графіків
    for (uint8_t k = 0; k < 120; k++)
    {
        chdr->tOutChart[k]   = 0;
        chdr->tInChart[k]    = 0;
        chdr->dTChart[k]     = 0;
        chdr->dT_pool_Chart[k]    = 0;
        chdr->tPoolChart[k]  = 0;
        chdr->V_Chart[k]     = 0;
        chdr->Q_Chart[k]     = 0;
```

```
     }

     chdr->CountLogs = 0;

// Генерація даних для демонстрації графіка отриманої енергії
    for (uint16_t k2 = 0; k2 < 280; k2++)
    {
        double phi = 2.0 * M_PI * (double)k2 * 2.0 / 280.0;   // 2
хвилі
        double s   = sin(phi);
        chdr->E_Chart[k2] = (uint16_t)(40 + (s + 1.0) * 60.0); //
масштабування 40..160
    }

    chdr->E_Chart[0] = 0;
     return chdr;
}

//оголошення вказівника на структуру CHART_DRY
CHART_DRY *chart_dry;    // структура  для графіків - інфо екрани 2
та 3. Значення за останні 24 години

//=================================================================
=======

//Оголошення функцій
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
     //обробник переривань таймера TIM1 раз на 3 секунди

bool FlashStorage_ReadSettings(DataForSave* settings);      //
Читання з flash структури settingRAM
bool FlashStorage_ReadChart(CHART_DRY* chart);                    //
Читання з flash структури chart_dry
bool FlashStorage_WriteSettings(DataForSave* settings);           //
запис на flash структури settingRAM
bool FlashStorage_WriteChart(CHART_DRY* chart);                   //
запис на flash структури chart_dry
static bool FlashStorage_Read(uint32_t start_addr, uint16_t
struct_size,
                      void* data, FlashMetadata* metadata);       //
Універсальна функція читання
static bool FlashStorage_Write(uint32_t start_addr, uint16_t
sector_count, uint16_t struct_size,
                               void* data, FlashMetadata* metadata);
     // Універсальна функція запису
bool FlashStorage_Init(SPI_HandleTypeDef *hspi, GPIO_TypeDef *gpio,
uint16_t pin,
                 CHART_DRY* chart, DataForSave* settings);  //
Ініціалізація
static bool UpdateMetadata(void);                                 //
Оновлення метаданих
static uint16_t FindLastMetadataPosition(void);                   //
Пошук останньої позиції метаданих
```

```
static bool IsSectorEmpty(uint8_t* buffer, uint16_t size);  //
Перевірка чистого сектору

void DEMO_ON(void); // увімкнути демо-режим
void DEMO_OFF(void); // вимкнути демо-режим
void show_time(void);              // відображення поточного часу,
дати
void get_time(int8_t F_update);    // отримання актуальних дати,
часу і формування масиву TimeBuf для виводу
void SetTime(void);                // встановлення заданих часу,
дати
void resetTimeDate(void);          // скидання дат мін/максимальних
статистичних значень
void writeSPIFlash();              // Запис на flash
void readSPIFlash();               // Читання даних з flash
void resetTemp(void);              // скидання min max значень
void dry_get_data(void);      // заповнення масиву графіка
void CheckEkran(void);             // перевірка чи не пора вимкнути
дисплей
void printSetTimeStatic(void);     // Екран "Налаштування Часу/Дати"
void Print_time_date(uint8_t *Dim_TimeDate, int x, int y, uint32_t
color);//форматування часу/дати у формат "12:34 12.12.17" і друк
вказаним кольором по вказаних координатах
void printSetTimeUpdate(void);          // Оновлення екрану
"Налаштування Часу/Дати"
void printSettingStatic(void);          //друк екрану налаштувань з
вибором режиму роботи

void printInfoStatic0(void);       // Друк статичної картинки екрана
0
void printInfoUpdate0(void);       // Друк значень на екрані 0
void printInfoStatic1(void);       // Друк статичної картинки екрана
1
void printInfoUpdate1(void);       // Друк значень на екрані 1
void printInfoStatic2(void);            // Друк статичної картинки
на інфо екрані 2
void printInfoUpdate2(void);       // Друк значень на інфо екрані 2
void printChart3(void);            // Друк графіка на інфо екрані 3
void updateChart3(void);      // Оновлення графіків на інфо екрані 3
void printHistoryLog4(void);       // Друк журнала історії на інфо
екрані 4
void UpdateHistoryLog4(void); // Оновлення журнала історії

void Print_period(uint32_t period, int x, int y, uint32_t color);
//форматування тривалості часу у формат "123дн 13год 21хв" і друк
вказаним кольором по вказаних координатах
char* Get_period(uint32_t period); //форматування тривалості часу у
формат "123дн 13год 21хв"

int8_t CheckON(void);              // Перевірка статусу системи, чи
не потрібно увімкнути/вимкнути зовнішні пристрої

void MotorON(void);           //увімкнення насоса колектора
```

```
void MotorOFF(void);          //вимкнення насоса колектора
void PumpPoolON();  //увімкнення насоса басейна
void PumpPoolOFF(); //вимкнення насоса басейна
void ValveON(void); //відкриття клапана
void ValveOFF(void);     //закриття клапана
void HeatON(void);            //увімкнення ТЕН
void HeatOFF(void);           //вимкнення ТЕН

void readNRF24(void);              //отримання параметрів з датчиків
int8_t button_click_handler(void); // обробник натискання кнопок
void Goto_New_Window(void);             //функція відображає новий
інфо екран

void printColorMode(void);              //Екран "Колірна схема"
void printOptionTEN(void);              // Екран "Налаштування ТЕН"
void printCor_dT(void);                 // Екран "Налаштування dT
колектора"
void printCor_dT_Update(void);          // Оновлення екрану
"Налаштування dT колектора"
void printCor_dT_pool(void);                 // Екран "Коригування
dT басейна"
void printCor_dT_pool_Update(void);          // Оновлення екрану
"Коригування dT басейна"

void printCor_Time(void);               // Екран "Налаштування
TimePeriod"
void printCor_Time_Update(void);   // Оновлення екрану "Налаштування
TimePeriod"
void print_Work_Pump(void);             //Екран "Робота насоса"
void print_Work_Pump_Update(void); // Оновлення екрану "Робота
насоса"

void print_T_valve(void);               // Екран "Коригування T для
запобіжного режиму"
void print_T_valve_Update(void);   // Оновлення екрану "Коригування
T для запобіжного режиму"

void printDemoMode(void);               //Екран "Демонстраційний
режим"

void get_data_E(void);// заповнення масиву графіка енергії (за
останні 280 днів)
void printChart_E(void); // Друк графіка отрманої енергії

static inline void packetToByteArray_RX(const Packet_RX *pkt,
uint8_t *byteArray);     // Функція перетворення структури в масив
байтів для приймання
static inline void BufferToPacket_RX(const uint8_t *buffer,
Packet_RX *out_pkt);     // Функція перетворення масива байтів в
структуру для приймання
```

```
static inline void packetToByteArray_TX(const Packet_TX *pkt,
uint8_t *byteArray);     // Перетворення структури в масив байтів
для передавання
static inline void BufferToPacket_TX(const uint8_t *buffer,
Packet_TX *out_pkt);     // Перетворення масиву байтів у вже існуючу
структуру для передавання

void printStartInfo(void);    //напис на екрані при старті

void print_Mode_PUMP_KOL(void);         // Екран "Вибір режиму
роботи насоса сонячного колектора"
void print_Mode_PUMP_POOL(void);        // Екран "Вибір режиму
роботи насоса басейна"
void print_Modes_T_Pool(void);               //Екран "Коригування
температур режимів"
void print_Modes_T_Pool_Update(void);   // Оновлення екрану "Робота
насоса"

void InsertIvent(int8_t  ID_IVENT);     //додавання ID події в
журнал історії
char* Get_Ivent(int8_t ID_IVENT);  //отримання опису події

//   - текстові масиви   ------------------------------------
// масив для форматування інформації перед виводом її на дисплей
(температура, вологість і тиск)
char TempBuf[127];

// масив для форматування інформації перед виводом її на дисплей
(дата, час)
char TimeBuf[15];

// масив використовується для перекодування тексту
char OutputBuf[127];

// масив для виводу строки з помилками
char ErrorBuf[127];

// - КОНСТАНТи ------------------------------------
// захист від замороження
#define TEMP_MIN         100       // Мінімально допустима
температура в теплообміннику (в сотих градуса) - вмикається
трубчастий електричний нагрівач (ТЕН)

// - ЧАСОВІ КОНСТАНТи -----------------------------------
#define TIME_SCAN_SENSOR 3         // Період опитування датчиків,
сек.
#define TIME_HOUR        3600      // Кількість сек. в 1 год
#define TIME_MIN         20        // константа для коректного
вираховування часу роботи системи
                                                 // = 60 сек. / 3
сек. (TIME_SCAN_SENSOR) = 20
```

```
int16_t TIME_PRINT_CHART = 720;              // Період виводу однієї
точки графіка, сек. 12 хв. = 720 сек.

//   для збереження ресурсу насосів і зменшення циклів запуску-
зупинки
uint32_t time_motor_ON = 0;   // час роботи насоса сонячного
колектора після його старта - вимірюється в інтервалах сканування
датчиків = TIME_SCAN_SENSOR
uint32_t time_motor_OFF = 0;// час зупину насоса сонячного колектора
- вимірюється в інтервалах сканування датчиків = TIME_SCAN_SENSOR
uint32_t time_motor_pool_ON = 0;   // час роботи насоса басейна
після його старта - вимірюється в інтервалах сканування датчиків =
TIME_SCAN_SENSOR
uint32_t time_motor_pool_OFF = 0;// час зупину насоса басейна -
вимірюється в інтервалах сканування датчиків = TIME_SCAN_SENSOR

uint32_t time_valve_ON = 0;   // час знаходження клапана скиду
горячої води у відкритому стані
uint32_t time_valve_OFF = 0;//у закритому


uint32_t time_heat_ON = 0;    // час роботи ТЕН після його
увімкнення - вимірюється в інтервалах сканування датчиків =
TIME_SCAN_SENSOR
uint32_t time_heat_OFF = 0;// час вимкнення ТЕН - вимірюється в
інтервалах сканування датчиків = TIME_SCAN_SENSOR

#define TIME_EKRAN       180       //час очікування натискання
кнопок, після якого підсвітка дисплея вимикається

int16_t hour_ekran =     0;        //час роботи дисплея після
останнього натискання кнопки

//--- режими роботи насоса сонячного колектора ----------------------
-
#define PUMP_KOL_OFF     1         // насос сонячного колектора
вимкнено, але присутня природня циркуляція
#define FLOW_ON               2         // Режим примусової
циркуляції при dT>0
#define FLOW_OFF         3         // Режим примусової циркуляції
при dT>8 (економічний режим)

//--- режими роботи ТЕН ----------------------
#define HEATING_OFF      1         // ТЕН не вмикається
#define HEATING_ON       2         // ТЕН повинен вмикатись в
потрібні часові інтервали якщо темп.в теплообміннику (Твх) меньше
заданої
#define HEATING_ALL      3         // ТЕН постійно підтримує
температуру

//--- режими роботи насоса басейна ----------------------
#define MODE_1_PUMP_POOL 1    //Спортивне плавання     24-26 °C
```

```
#define MODE_2_PUMP_POOL 2    //Відпочинок, купання дорослих     26-
28 °C
#define MODE_3_PUMP_POOL 3    //Діти, сімейне купання  28–30 °C
#define MODE_4_PUMP_POOL 4    //Терапевтичні або спа-цілі   30–32 °C
#define MODE_5_PUMP_POOL 5    //Користувацькі налаштування
#define MODE_6_PUMP_POOL 6    //Насос басейна вимкнено

#define NUM_INFO_SCREEN       5         // Кількість інформаційних
екранів
#define NUM_POS                  14         // Кількість позицій на
екрані налаштувань
#define NUM_SCREEN           51         // Кількість екранів

#define dist   8+14                //для дистанціювання рядків при
виводі:наприклад, dist*5 замість 8+14*5

int8_t packet_ready      =    0;        // 1 -  оброблено нові дані
з датчиків

uint8_t last_min = 200;             // для виводу часу
uint8_t last_day = 200;             // для виводу дати

LCD_Handler *lcd;              // оголошуємо вказівник на дисплей

//CalibrationParams calib;

// тип визначає колірну схему
typedef struct Color_Scheme
{                // кольори:
     uint32_t text_color_dir,       // -тексту найменування каталогу
     text_color_file,          // -тексту найменування файлу
     text_color_cursor,             // -тексту вибраного
файлу/каталогу
     bg_color,                 // -фону оточення
     bg_line1_color,           // -фону парних рядків
     bg_line2_color,           // -фону непарних рядків
     cursor_color,             // -курсора на поточному
файлі/каталозі
     slider_color;             // -повзунка прокручування
} Color_Scheme;

Color_Scheme *color_scheme[4];// Оголошення масиву вказівників на
колірні схеми

File_Manager_Handler *fm;          // оголошуємо обробника файлового
менеджера для підтримки колірних схем

#define PUMP_BIT         0            // біт увімкнення насоса
сонячного колектора в settingRAM->flags
#define HEAT_BIT         1            // біт увімкнення ТЕН в
settingRAM->flags
#define PUMPPOOL_BIT     2            // біт увімкнення насоса
басейна
```

```
#define VALVE_BIT        3            // біт відкриття клапана
#define DEMO_BIT         4            // біт увімкнення
демонстраційного режиму
#define EKRAN_BIT        5            // біт увімкнення дисплея


#define SET_FLAG(bit)         do {SET_BIT(settingRAM->flags, (1u <<
bit));} while (0)   // біт встановити в 1
#define CLEAR_FLAG(bit)       do {CLEAR_BIT(settingRAM->flags, (1u
<< bit));} while (0)     // біт встановити в 0

#define FLAG_PUMP_CHECK       ((settingRAM->flags & (1u<<PUMP_BIT))
!= 0)    // біт насоса перевірити на 1
#define MOTOR_ON()            SET_FLAG(PUMP_BIT)       // увімкнути
насос
#define MOTOR_OFF()           CLEAR_FLAG(PUMP_BIT)     // вимкнути
насос

#define FLAG_HEAT_CHECK       ((settingRAM->flags & (1u<<HEAT_BIT))
!= 0)    // біт нагрівача перевірити на 1
#define HEAT_ON()        SET_FLAG(HEAT_BIT)  // увімкнути ТЕН
#define HEAT_OFF()       CLEAR_FLAG(HEAT_BIT)     // вимкнути ТЕН

#define FLAG_PUMPPOOL_CHECK   ((settingRAM->flags &
(1u<<PUMPPOOL_BIT)) != 0)    // біт насоса басейна перевірити на 1
#define PUMPPOOL_ON()         SET_FLAG(PUMPPOOL_BIT)   // увімкнути
насос
#define PUMPPOOL_OFF()        CLEAR_FLAG(PUMPPOOL_BIT) // вимкнути
насос

#define FLAG_VALVE_CHECK ((settingRAM->flags & (1u<<VALVE_BIT)) !=
0)    // біт  відкриття клапана перевірити на 1
#define VALVE_ON()       SET_FLAG(VALVE_BIT) // відкрити клапан
#define VALVE_OFF()      CLEAR_FLAG(VALVE_BIT)    // закрити клапан

#define FLAG_EKRAN_CHECK ((settingRAM->flags & (1u<<EKRAN_BIT)) !=
0)    // біт дисплея перевірити на 1
#define EKRAN_ON()       do { LCD_SleepOut(lcd);
SET_FLAG(EKRAN_BIT); hour_ekran    = 0;} while (0)     //виведення
дисплея зі сплячого режиму (увімкнення відображення дисплея та
підсвічування)
#define EKRAN_OFF()      do { LCD_SleepIn(lcd);
CLEAR_FLAG(EKRAN_BIT); } while (0)      //переведення дисплея в
"сплячий режим" (вимкнення відображення дисплея та підсвічування)

#define FLAG_DEMO_CHECK  ((settingRAM->flags & (1u<<DEMO_BIT)) != 0)
// біт демо-режиму перевірити на 1
// Період виводу однієї точки графіка, сек. 12 хв. = 720 сек.
// Період виводу однієї точки графіка, в DEMO одна точка кожні 3
сек.

// структура Packet для збереження поточних значень
typedef struct Packet
```

```
{
     int16_t        tOut;          // поточна температура на вході в
теплообмінник - в сотих градуса
     int16_t        tIn;      // поточна температура на виході з
теплообмінника - в сотих градуса
     int16_t        tPool;         // поточна температура в басейні
- в сотих градуса
     int16_t        dT;       // різниця температур колектра - в
сотих градуса
     int16_t        dT_pool;       // різниця температур басейна - в
сотих градуса
     uint16_t  V;        // витрата - в сотих літр на хвилину,
тільки позитивні значення
     uint16_t  Q_3s;          // Отримана енергія  за 3 секунди (в
сотих долях)
     uint16_t  Q_12m;         // Отримана енергія  за період виводу
точки на графік: 12 хв (в сотих долях)
}    Packet;

//функція задання початкових значень структури Packet
Packet* PacketNew(void)
{
     Packet *pct = (Packet*)calloc(1, sizeof(Packet));
     pct->tOut = 2500;   //+25 °C
     pct->tIn  = 2500;   //+25 °C
     pct->tPool     = 2500;   //+25 °C
     pct->dT        = 0; //+25 °C
     pct->dT_pool   = 0; //+25 °C
     pct->V         = 0; //0 л/с
     pct->Q_3s      = 0; //0 кДж
     pct->Q_12m          = 0; //0 кДж
     return pct;
}

Packet *packet; //оголошення вказівника на структуру Packet під
ім’ям packet

struct KILCE;  // Структура KILCE для навігації по меню

//оголошення функцій для роботи зі структурою KILCE
typedef void (*KilceSet_Poz)(struct KILCE *klc, int16_t
Poz);//встановлення поточного значення позиції
typedef int16_t (*KilceGet_Poz)(struct KILCE *klc);    //отримання
поточного значення позиції
typedef void (*KilceUp)(struct KILCE *klc);       //збільшення
позиції
typedef void (*KilceDown)(struct KILCE *klc);          //зменшення
позиції

typedef struct KILCE
{
     int16_t _Min;       // мінімальне значення позиції
     int16_t _Max;       // максимальне значення позиції
```

```
     int16_t _Poz;       // поточне значення позиції
     KilceSet_Poz Set_Poz;    // встановлення поточного значення
позиції
     KilceGet_Poz Get_Poz;    // отримання поточного значення
позиції
     KilceUp Up;         // збільшення позиції
     KilceDown Down;          // зменшення позиції
}    KILCE;

static inline void DoSetPoz(KILCE *klc, int16_t Poz)
     //встановлення поточного значення позиції
{
     if(Poz > klc->_Max) klc->_Poz = klc->_Max;
          else if (Poz < klc->_Min) klc->_Poz = klc->_Min;
               else klc->_Poz = Poz;
}

static inline int16_t DoGetPoz(KILCE *klc)        //отримання
поточного значення позиції
{    return klc->_Poz;   }

static inline void DoUp(KILCE *klc)               //збільшення
позиції
{    klc->_Poz >= klc->_Max ? klc->_Poz = klc->_Min : klc->_Poz++;
     }

static inline void DoDown(KILCE *klc)             //зменшення
позиції
{    klc->_Poz <= klc->_Min ? klc->_Poz = klc->_Max : klc->_Poz--;
     }

KILCE* KILCE_New(int16_t Min, int16_t Max)   //функція задання
початкових значень структури KILCE
{
     KILCE *klc     = (KILCE*)calloc(1, sizeof(KILCE));
     klc->_Min = Min;
     klc->_Max = Max;
     klc->_Poz = Min;
     klc->Set_Poz   = DoSetPoz;
     klc->Get_Poz   = DoGetPoz;
     klc->Up        = DoUp;
     klc->Down = DoDown;
     return klc;
}

//оголошення вказівників на структури KILCE
KILCE *ScreenMode;       //для збереження номера поточного меню
KILCE *Screens[NUM_SCREEN];   //для збереження позиції в меню

// Структура AVARAGE для визначення середніх значень (значення
ковзного середнього)
// середньє арифметичне не застосовується, т.я. для нього потрібно
виділяти багато пам'яті
```

```
// застосовувати середнє значення необхідно для зглажування
пульсацій вимірювань
struct AVARAGE;

//оголошення функцій для роботи зі структурою AVARAGE
typedef void (*AveragePush)(struct AVARAGE *avrg, int16_t X);//ф-я
додає чергове значення і обчислюється ковзне середнє
typedef int16_t (*AverageMean)(struct AVARAGE *avrg);  //функція
вертає значення ковзного середнього
typedef void (*AverageErase)(struct AVARAGE *avrg);

typedef struct AVARAGE
{
     int32_t _sum;        // сума всіх значень
     uint8_t _MaxSize;    // максимальна кількість значень
     uint8_t _count;      // поточна кількість значень
     int16_t _mean;       // ковзне середнє значення
     AveragePush Push;         // функція додає чергове значення і
обчислюється ковзне середнє
     AverageMean Mean;         // функція вертає значення ковзного
середнього
     AverageErase Erase;       //функція видаляє значення
}    AVARAGE;

static inline int16_t GetMean(AVARAGE *avrg) //функція вертає
значення ковзного середнього
{    return avrg->_mean; }

void DoPush(AVARAGE *avrg, int16_t X)   //функція додає чергове
значення і обчислюється ковзне середнє
{
     if(avrg->_count >= avrg->_MaxSize)
     {
          avrg->_sum =   (int32_t)(avrg->_sum - avrg->_mean + X);
          avrg->_mean =  (int16_t)(avrg->_sum / avrg->_MaxSize);
     }
     else
     {
          if(avrg->_count == 0)
          { avrg->_sum = X; avrg->_count = 1; avrg->_mean = X; }
          else
          {
               avrg->_count++;
               avrg->_sum = (int32_t)(avrg->_sum + X);
               avrg->_mean = (int16_t)(avrg->_sum / avrg->_count);
          }
     }
}

void DoErase(AVARAGE *avrg)   //функція видаляє значення
{ avrg->_count = 0; }
```

```
AVARAGE* AVARAGE_New(uint8_t size) //функція задання початкових
значень структури AVARAGE
{
     AVARAGE *avrg       = (AVARAGE*)calloc(1, sizeof(AVARAGE));
     avrg->_MaxSize = size;
     avrg->_count = 0;
     avrg->_sum     = 0;
     avrg->_mean = 0;
     avrg->Push     = DoPush;
     avrg->Mean     = GetMean;
     avrg->Erase    = DoErase;
     return avrg;
}

//оголошення вказівників на структури AVARAGE
//для обчислення та збереження середніх значень показів датчиків
AVARAGE *ave_tIn;
AVARAGE *ave_tOut;
AVARAGE *ave_tPool;
AVARAGE *ave_V;

//для приведення показів датчиків до одного рівня
AVARAGE *ave_tOutK;


#ifdef  USE_ADC
  AVARAGE *ave_t_MK;
  AVARAGE *ave_V_MK;
#endif

void print_error(char *txt_error)
{
     if(system_ready == 0) return;
     LCD_WriteString(lcd, 0, 190, utf8_to_win1251(txt_error,
OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor, fm-
>color_scheme->cursor_color, LCD_SYMBOL_PRINT_FAST);
}

void test(void)
{
     uint8_t poz;
     poz = ScreenMode->Get_Poz(ScreenMode);
     sprintf(TempBuf, "SM=%d S=%d", poz, Screens[poz]-
>Get_Poz(Screens[poz]));

     LCD_WriteString(lcd, 240, 17, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
}

void test2(void)
{
  uint8_t value = chart_dry->CountLogs;
```

```
  sprintf(TempBuf, "CL = %d", value);
  LCD_WriteString(lcd, 240, 17, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
}

//напис на екрані:
//Комп’ютеризована система управління сонячним колектором на базі
STM32
//Кваліфікаційна робота на здобуття магістра Віталія Невмержицького
void printStartInfo(void)
{
     if(Start_Info_time == 2)
     {
          LCD_Fill(lcd, COLOR_BLACK);
          LCD_WriteString(lcd, 10, 0, utf8_to_win1251("Методи та
засоби контролю", OutputBuf), &Font_15x25, COLOR_YELLOW,
COLOR_BLACK, LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 10, 30, utf8_to_win1251("сонячним
колектором для", OutputBuf), &Font_15x25, COLOR_YELLOW, COLOR_BLACK,
LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 10, 60, utf8_to_win1251("управління
температурою", OutputBuf), &Font_15x25, COLOR_YELLOW, COLOR_BLACK,
LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 10, 90, utf8_to_win1251("води в
басейні", OutputBuf), &Font_15x25, COLOR_YELLOW, COLOR_BLACK,
LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 40, 150,
utf8_to_win1251("Кваліфікаційна робота", OutputBuf), &Font_15x25,
COLOR_YELLOW, COLOR_BLACK, LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 40, 180, utf8_to_win1251("на здобуття
магістра", OutputBuf), &Font_15x25, COLOR_YELLOW, COLOR_BLACK,
LCD_SYMBOL_PRINT_FAST);
          LCD_WriteString(lcd, 50, 210, utf8_to_win1251("Віталія
Невмержицького", OutputBuf), &Font_15x25, COLOR_YELLOW, COLOR_BLACK,
LCD_SYMBOL_PRINT_FAST);
     }
     Start_Info_time--;
     if(Start_Info_time == 0) Goto_New_Window();  // Друк поточного
інфо екрана
}

// Функція перетворення структури в масив байтів
static inline void packetToByteArray_RX(const Packet_RX *pkt,
uint8_t *byteArray) {
  // Записуємо температуру T1 (2 байти)
    byteArray[0] = (uint8_t)(pkt->t1 & 0xFF);        // молодший
байт
    byteArray[1] = (uint8_t)((pkt->t1 >> 8) & 0xFF); // старший байт

    // Записуємо температуру T2 (2 байти)
    byteArray[2] = (uint8_t)(pkt->t2 & 0xFF);
    byteArray[3] = (uint8_t)((pkt->t2 >> 8) & 0xFF);
```

```
    // Записуємо температуру T3 (2 байти)
    byteArray[4] = (uint8_t)(pkt->t3 & 0xFF);
    byteArray[5] = (uint8_t)((pkt->t3 >> 8) & 0xFF);

    // Записуємо кількість імпульсів датчика витрати за останні три
секунди (2 байти)
    byteArray[6] = (uint8_t)(pkt->V & 0xFF);
    byteArray[7] = (uint8_t)((pkt->V >> 8) & 0xFF);
}

// Функція перетворення масива байтів в структуру
static inline void BufferToPacket_RX(const uint8_t *buffer,
Packet_RX *out_pkt){
    // Перетворення little-endian байтів у int16_t
    // t1 (температура): перші 2 байти
    out_pkt->t1 = (int16_t)((buffer[1] << 8) | buffer[0]);

    // t2 (температура): наступні 2 байти
    out_pkt->t2 = (int16_t)((buffer[3] << 8) | buffer[2]);

    // t3 (температура): наступні 2 байти
    out_pkt->t3 = (int16_t)((buffer[5] << 8) | buffer[4]);

    // кількість імпульсів датчика витрати за останні три секунди:
останні 2 байти
    out_pkt->V  = (int16_t)((buffer[7] << 8) | buffer[6]);
}

// Перетворення структури в масив байтів
static inline void packetToByteArray_TX(const Packet_TX *pkt,
uint8_t *byteArray) {
    // просто копіюємо всю пам’ять структури
    memcpy(byteArray, pkt, PLD_SIZE);
}

// Перетворення масиву байтів у вже існуючу структуру.
// out_pkt повинен бути валідним покажчиком на Packet_TX.
static inline void BufferToPacket_TX(const uint8_t *buffer,
Packet_TX *out_pkt) {
    memcpy(out_pkt, buffer, PLD_SIZE);
}

void DEMO_ON(void) // увімкнути демо-режим
{
     SET_FLAG(DEMO_BIT);
     Screens[10]->Set_Poz(Screens[10], 1);
     settingRAM->demo_mode = 1;
     TIME_PRINT_CHART = 3;
     HEAT_OFF(); time_heat_OFF = 0; time_heat_ON = 0;
     MOTOR_OFF(); time_motor_OFF = 0; time_motor_ON = 0;
     PUMPPOOL_OFF(); time_motor_pool_OFF = 0; time_motor_pool_ON =
0;
```

```
	VALVE_OFF(); time_valve_OFF = 0; time_valve_ON = 0;

	if(system_ready == 1)
	{
		ave_tPool->Erase(ave_tPool);
		ave_tIn->Erase(ave_tIn);
		ave_tOut->Erase(ave_tOut);
		ave_V->Erase(ave_V);
		ave_tIn->Push(ave_tIn, 2300);
		ave_tOut->Push(ave_tOut, 2300);
		ave_tPool->Push(ave_tPool, 2300);
		ave_V->Push(ave_V, 0);
		packet->dT_pool = 0;
		packet->dT = 0;
		InsertIvent(9);
	}
}

void DEMO_OFF(void) // вимкнути демо-режим
{
	CLEAR_FLAG(DEMO_BIT);
	Screens[10]->Set_Poz(Screens[10], 0);
	settingRAM->demo_mode = 0;
	TIME_PRINT_CHART = 720;

	HEAT_OFF();
	MOTOR_OFF();
	PUMPPOOL_OFF();
	VALVE_OFF();

	if(system_ready == 1)
	{
		time_heat_OFF =  1;
		time_heat_ON = 0;

		time_motor_OFF =  1;
		time_motor_ON = 0;

		time_motor_pool_OFF = 1;
		time_motor_pool_ON = 0;

		time_valve_OFF = 0;
		time_valve_ON = 0;

		ave_tIn->Erase(ave_tIn);
		ave_tOut->Erase(ave_tOut);
		ave_tPool->Erase(ave_tPool);
		ave_V->Erase(ave_V);

		ave_tIn->Push(ave_tIn, 2300);
		ave_tOut->Push(ave_tOut, 2300);
		ave_tPool->Push(ave_tPool, 2300);
		ave_V->Push(ave_V, 0);
```

```
          packet->dT_pool = 0;
          packet->dT = 0;

          InsertIvent(10);
     }
}

void show_time(void)     //відображення поточного часу
{
     LCD_FillWindow(lcd, 0, 0, 319, 14, fm->color_scheme-
>cursor_color);
     LCD_WriteString(lcd, 207, 1, TimeBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);

     if(FLAG_DEMO_CHECK)
     {    LCD_WriteString(lcd, 15, 1,
utf8_to_win1251("демонстраційний режим", OutputBuf), &Font_12x20,
COLOR_RED, fm->color_scheme->cursor_color, LCD_SYMBOL_PRINT_FAST); }
    else
    if (ErrorBuf[0] != 0)
    { LCD_WriteString(lcd, 15, 1, utf8_to_win1251(ErrorBuf,
OutputBuf), &Font_12x20, COLOR_RED, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST); }
}

//отримання дати, часу і формування масиву TimeBuf для виводу
void get_time(int8_t F_update)//Якщо F_update=0, то оновлення
TimeBuf тільки при зміні хвилини
{
     HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);        //
отримуємо час у форматі RTC_FORMAT_BIN
     HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);        //
отримуємо дату у форматі RTC_FORMAT_BIN

     if (last_min != sTime.Minutes || F_update == 1)  // виводимо
тільки зміну - тобто, раз в хвилину. Або якщо встановлено ознаку
примусового оновлення F_update == 1
     {
          last_min = sTime.Minutes;
          // формуємо строку для виводу часу і дати
          last_min = sTime.Hours;
          if (last_min < 10) TimeBuf[0] = 48; else TimeBuf[0] =
last_min / 10 + 48;   // код "0" 48
          TimeBuf[1] = last_min % 10 + 48;
          TimeBuf[2] = 58; // ":"
          last_min = sTime.Minutes;
          if (last_min < 10) TimeBuf[3] = 48; else TimeBuf[3] =
last_min / 10 + 48;
          TimeBuf[4] = last_min % 10 + 48;
          if (last_day != sDate.Date || F_update == 1)
          {
```

```
                if (last_day != sDate.Date)
                {
                     if(chart_dry->posDayE != sDate.Date &&
chart_dry->posDayE != 0)
                     {
                          chart_dry->posDayE = sDate.Date;
                          get_data_E();
                     }
                }
                TimeBuf[5] = 32;// " "
                last_day = sDate.Date;
                if (last_day < 10) TimeBuf[6] = 48; else TimeBuf[6] =
last_day / 10 + 48;
                TimeBuf[7] = last_day % 10 + 48;
                TimeBuf[8] = 46;// "."

                uint8_t temp;
                temp = sDate.Month;
                if (temp < 10) TimeBuf[9] = 48; else TimeBuf[9] =
temp / 10 + 48;
                TimeBuf[10] = temp % 10 + 48;
                TimeBuf[11] = 46;// "."
                temp = sDate.Year;
                if (temp < 10) TimeBuf[12] = 48; else TimeBuf[12] =
temp / 10 + 48;
                TimeBuf[13] = temp % 10 + 48;
                TimeBuf[14] = 0; // Кінець строки
           }
     }
}

void SetTime(void) // встановлення заданих часу, дати
{
     LCD_Fill(lcd, fm->color_scheme->bg_color);
     sTime.Hours = Screens[2]->Get_Poz(Screens[2]);
     sTime.Minutes = Screens[3]->Get_Poz(Screens[3]);
     sTime.Seconds = 0;
     sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
     sTime.StoreOperation = RTC_STOREOPERATION_RESET;

     HAL_StatusTypeDef rez;
     rez = HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
     if (rez == HAL_OK)
          LCD_WriteString(lcd, 10, 30, utf8_to_win1251("Час успішно
змінено!", OutputBuf), &Font_15x25, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     else
          switch (rez)
          {
               case  HAL_ERROR:
                     LCD_WriteString(lcd, 1, 30,
utf8_to_win1251("HAL_ERROR", OutputBuf), &Font_15x25, fm-
```

```
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
                        break;
                case  HAL_BUSY:
                        LCD_WriteString(lcd, 1, 30,
utf8_to_win1251("HAL_BUSY", OutputBuf), &Font_15x25, fm-
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
                        break;

                case  HAL_TIMEOUT:
                        LCD_WriteString(lcd, 1, 30,
utf8_to_win1251("HAL_TIMEOUT", OutputBuf), &Font_15x25, fm-
>color_scheme->text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
                        break;
                default:
                  LCD_WriteString(lcd, 10, 30, utf8_to_win1251("Час
не вдалося змінити!", OutputBuf), &Font_15x25, fm->color_scheme-
>text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
          }

     sDate.Month = Screens[5]->Get_Poz(Screens[5]);
     sDate.Date = Screens[4]->Get_Poz(Screens[4]);
     sDate.Year = Screens[6]->Get_Poz(Screens[6]);

     if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BIN) == HAL_OK)
     {
          LCD_WriteString(lcd, 10, 60, utf8_to_win1251("Дату успішно
змінено!", OutputBuf), &Font_15x25, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
          writeSPIFlash();    // Запис на flash
     }
     else
          LCD_WriteString(lcd, 10, 60, utf8_to_win1251("Дату не
вдалося змінити!", OutputBuf), &Font_15x25, fm->color_scheme-
>text_color_cursor, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);

     HAL_Delay(1000);

     get_time(1);   // отримання актуальних дати, часу і формування
масиву TimeBuf для виводу
}

void resetTimeDate(void) //скидання дат мін/максимальних
статистичних значень
{
     uint8_t   var;

     for(int i=0;i<5;i++)
```

```
     {
          switch (i)
          {
               case  0:                  //minute
               case  1:  var = 0;  break;    //hour
               case  2:                  //day
               case  3:  var = 11; break;    //month
               case  4:  var = 25; break;    //year
          }
          settingRAM->tOutMinTimeDate[i] = var;
          settingRAM->tOutMaxTimeDate[i] = var;
          settingRAM->tInMinTimeDate[i] = var;
          settingRAM->tInMaxTimeDate[i] = var;
          settingRAM->tPoolMinTimeDate[i] = var;
          settingRAM->tPoolMaxTimeDate[i] = var;
          settingRAM->SaveTimeDate[i] = var;
     }
}

// Запис на flash ------------------------------------------------
void writeSPIFlash(void)
{
     get_time(1);
     // оновлення часу останнього збереження
     settingRAM->SaveTimeDate[0] = sTime.Minutes;
     settingRAM->SaveTimeDate[1] = sTime.Hours;
     settingRAM->SaveTimeDate[2] = sDate.Date;
     settingRAM->SaveTimeDate[3] = sDate.Month;
     settingRAM->SaveTimeDate[4] = sDate.Year;

     tick_SPI_Flash = 0;
     FlashStorage_WriteSettings(settingRAM);// запис на flash
структури settingRAM
     FlashStorage_WriteChart(chart_dry);     // запис на flash
структури chart_dry
     FlashStorage_Deinit();
}

// Читання даних з flash ---------------------------------------------
-------
void readSPIFlash(void)
{
     if(!FlashStorage_ReadSettings(settingRAM)){  // Читання з flash
структури settingRAM
          settingRAM = DataForSaveNew();
     }
     if(!FlashStorage_ReadChart(chart_dry)){      // Читання з flash
структури chart_dry
          chart_dry = CHART_DRY_New();
     }
     Screens[0]->Set_Poz(Screens[0], settingRAM->N_Screen);//
встановлюємо поточним інфо екран, зчитаний з flash
```

```
     Screens[1]->Set_Poz(Screens[1], 1);// позиція в меню
налаштувань
     Screens[2]->Set_Poz(Screens[2], sTime.Hours);
     Screens[3]->Set_Poz(Screens[3], sTime.Minutes);
     Screens[4]->Set_Poz(Screens[4], sDate.Date);
     Screens[5]->Set_Poz(Screens[5], sDate.Month);
     Screens[6]->Set_Poz(Screens[6], sDate.Year);
     Screens[7]->Set_Poz(Screens[7], 0);//8 - Так/Ні

     Screens[9]->Set_Poz(Screens[9], settingRAM->color_mode);
     Screens[10]->Set_Poz(Screens[10], settingRAM-
>demo_mode);//Вимкнено / Увімкнено демонстраційний режим
     Screens[11]->Set_Poz(Screens[11], settingRAM-
>mode_flow);//режим роботи насоса сонячного колектора
     Screens[12]->Set_Poz(Screens[12], settingRAM-
>calibrationRTC);//"Коригування ходу годинника"

     Screens[13]->Set_Poz(Screens[13], settingRAM-
>T_max_valve);//T_max_valve
     Screens[14]->Set_Poz(Screens[14], settingRAM-
>mode_heat);//режим роботи ТЕН

//   Налаштування таблиці застосування ТЕН
     Screens[16]->Set_Poz(Screens[16], settingRAM->Use_Period_1);
     Screens[17]->Set_Poz(Screens[17], settingRAM->PeriodTime_1[0]);
     Screens[18]->Set_Poz(Screens[18], settingRAM->PeriodTime_1[1]);
     Screens[19]->Set_Poz(Screens[19], settingRAM->PeriodTime_1[2]);
     Screens[20]->Set_Poz(Screens[20], settingRAM->PeriodTime_1[3]);

     Screens[21]->Set_Poz(Screens[21], settingRAM->Use_Period_2);
     Screens[22]->Set_Poz(Screens[22], settingRAM->PeriodTime_2[0]);
     Screens[23]->Set_Poz(Screens[23], settingRAM->PeriodTime_2[1]);
     Screens[24]->Set_Poz(Screens[24], settingRAM->PeriodTime_2[2]);
     Screens[25]->Set_Poz(Screens[25], settingRAM->PeriodTime_2[3]);

     Screens[26]->Set_Poz(Screens[26], settingRAM->Use_Period_3);
     Screens[27]->Set_Poz(Screens[27], settingRAM->PeriodTime_3[0]);
     Screens[28]->Set_Poz(Screens[28], settingRAM->PeriodTime_3[1]);
     Screens[29]->Set_Poz(Screens[29], settingRAM->PeriodTime_3[2]);
     Screens[30]->Set_Poz(Screens[30], settingRAM->PeriodTime_3[3]);
     Screens[31]->Set_Poz(Screens[31], 0);   //   Зберегти/Скасувати

     Screens[32]->Set_Poz(Screens[32], settingRAM-
>dT_min_pump);//поріг увімкнення насоса колектора по різниці
температур в градусах, тільки позитивні значення
     Screens[33]->Set_Poz(Screens[33], settingRAM->dT_OFF); //
Гістерезис температури в градусах
     Screens[34]->Set_Poz(Screens[34], settingRAM->MIN_ON_TIME); //
мінімальна тривалість роботи насоса у хвилинах
     Screens[35]->Set_Poz(Screens[35], settingRAM->MIN_OFF_TIME);
     // мінімальна тривалість відпочинку насоса у хвилинах
     Screens[36]->Set_Poz(Screens[36], 0);   //   Зберегти/Скасувати
```

```
     Screens[37]->Set_Poz(Screens[37], 0);        //графік за
останні 280 днів

     Screens[38]->Set_Poz(Screens[38], settingRAM->mode_pump_pool);
          //режим насоса басейна
     Screens[39]->Set_Poz(Screens[39], settingRAM->T_min_pool[0]);
     //min температура для режиму 1 басейна
     Screens[40]->Set_Poz(Screens[40], settingRAM->T_max_pool[0]);
     //max температура для режиму 1 басейна
     Screens[41]->Set_Poz(Screens[41], settingRAM->T_min_pool[1]);
     //min температура для режиму 2 басейна
     Screens[42]->Set_Poz(Screens[42], settingRAM->T_max_pool[1]);
     //max температура для режиму 2 басейна
     Screens[43]->Set_Poz(Screens[43], settingRAM->T_min_pool[2]);
     //min температура для режиму 3 басейна
     Screens[44]->Set_Poz(Screens[44], settingRAM->T_max_pool[2]);
     //max температура для режиму 3 басейна
     Screens[45]->Set_Poz(Screens[45], settingRAM->T_min_pool[3]);
     //min температура для режиму 4 басейна
     Screens[46]->Set_Poz(Screens[46], settingRAM->T_max_pool[3]);
     //max температура для режиму 4 басейна
     Screens[47]->Set_Poz(Screens[47], settingRAM->T_min_pool[4]);
     //min температура для режиму 5 басейна
     Screens[48]->Set_Poz(Screens[48], settingRAM->T_max_pool[4]);
     //max температура для режиму 5 басейна
     Screens[49]->Set_Poz(Screens[49], 0);   //   Зберегти/Скасувати

     Screens[50]->Set_Poz(Screens[50], settingRAM-
>dT_min_pump_pool);//поріг увімкнення насоса басейна по різниці
температур в градусах, тільки позитивні значення
}

void resetTemp(void)  // скидання min max значень
{
     time_heat_OFF = 0;
     time_heat_ON = 0;
     time_motor_OFF = 0;
     time_motor_ON = 0;
     time_motor_pool_OFF = 0;
     time_motor_pool_ON = 0;
     time_valve_OFF = 0;
     time_valve_ON = 0;
     settingRAM = DataForSaveNew();
     chart_dry = CHART_DRY_New();
     writeSPIFlash();  //Скинули дані - зберігаємо на flash
     readSPIFlash();     //завантажуємо нові дані з параметрами по
замовчуванню
}

void printInfoStatic2()  // Друк статичної картинки на інфо екрані 2
{
// Таблиця
```

```
     LCD_DrawLineH(lcd, 0, 137, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 138, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 139, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 159, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 179, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 198, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 199, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 200, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 219, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, 239, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineV(lcd, 220,139,60, fm->color_scheme->slider_color);
     LCD_DrawLineV(lcd, 80,199,40, fm->color_scheme->slider_color);
     LCD_DrawLineV(lcd, 160,199,40, fm->color_scheme->slider_color);
     LCD_DrawLineV(lcd, 240,199,40, fm->color_scheme->slider_color);

  // Заголовки таблиць
     LCD_WriteString(lcd, 14, 143, utf8_to_win1251("T басейна, °C",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, fm-
>color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 14, 163, utf8_to_win1251("Витрата в
колекторі, л/хв", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 14, 183, utf8_to_win1251("Енергія за добу,
кВт·год", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 14, 203, utf8_to_win1251("T вх,°C",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, fm-
>color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 88, 203, utf8_to_win1251("T вих,°C",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, fm-
>color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 171, 203, utf8_to_win1251("dT кол",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, fm-
>color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 254, 203, utf8_to_win1251("dT бас",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, fm-
>color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
}

void printInfoUpdate2(void) // Друк значень на інфо екрані 2
{
```

```
     show_time();   //відображення поточного часу
     int8_t dst = 15;
     uint32_t col_z = fm->color_scheme->text_color_file;    //колір
букв
     uint32_t col_b = fm->color_scheme->text_color_cursor;  //колір
букв
     uint32_t col_f = fm->color_scheme->bg_color;           //колір
фону

//радіоканал
     if(error_RX>1)
     {
          sprintf(TempBuf, "Радіоканал: зв`язок відсутній  ");
          LCD_WriteString(lcd, 1, dst, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     }
     else
     {
          sprintf(TempBuf, "Радіоканал: зв`язок встановлено");
          LCD_WriteString(lcd, 1, dst, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);
     }

//насос
     if (FLAG_PUMPPOOL_CHECK)
     {
          sprintf(TempBuf, "Насос басейна: працює   ");
          LCD_WriteString(lcd, 1, dst*2, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     }
     else
     {
          sprintf(TempBuf, "Насос басейна: не працює");
          LCD_WriteString(lcd, 1, dst*2, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);
     }

     switch (settingRAM->mode_pump_pool)
     {
          case  MODE_1_PUMP_POOL:
               sprintf(TempBuf, "Режим 1: Спортивне плавання: %d-%d
°C   ", settingRAM->T_min_pool[0], settingRAM->T_max_pool[0]);
               break;
          case  MODE_2_PUMP_POOL:
               sprintf(TempBuf, "Режим 2: Купання дорослих: %d-%d °C
", settingRAM->T_min_pool[1], settingRAM->T_max_pool[1]);
               break;
          case  MODE_3_PUMP_POOL:
               sprintf(TempBuf, "Режим 3: Діти, сімейне купання: %d-
%d °C", settingRAM->T_min_pool[2], settingRAM->T_max_pool[2]);
               break;
          case  MODE_4_PUMP_POOL:
```

```
                sprintf(TempBuf, "Режим 4: Терапевтичні цілі: %d-%d
°C    ", settingRAM->T_min_pool[3], settingRAM->T_max_pool[3]);
                break;
           case  MODE_5_PUMP_POOL:
                sprintf(TempBuf, "Режим 5: Користувацький режим: %d-
%d °C ", settingRAM->T_min_pool[4], settingRAM->T_max_pool[4]);
                break;
           case  MODE_6_PUMP_POOL:
                sprintf(TempBuf, "Режим 6: Насос басейна вимкнено
");
                break;
     }
     LCD_WriteString(lcd, 1, dst*3, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);

//насос сонячного колектора
     if (FLAG_PUMP_CHECK)
     {
          sprintf(TempBuf, "Насос колектора: працює   ");
          LCD_WriteString(lcd, 1, dst*4, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     }
     else
     {
          sprintf(TempBuf, "Насос колектора: не працює");
          LCD_WriteString(lcd, 1, dst*4, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);
     }

     switch (settingRAM->mode_flow)
     {
          case  PUMP_KOL_OFF:
                sprintf(TempBuf, "Режим 1: Насос вимкнено        ");
                break;
          case  FLOW_ON:
                sprintf(TempBuf, "Режим 2: Циркуляція при dT>0°C ");
                break;
          case  FLOW_OFF:
                sprintf(TempBuf, "Режим 3: Циркуляція при dT>%d°C",
settingRAM->dT_min_pump);
                break;
     }
     LCD_WriteString(lcd, 1, dst*5, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);

// ТЕН
     if (FLAG_HEAT_CHECK)
     {
          sprintf(TempBuf, "ТЕН: працює   ");
          LCD_WriteString(lcd, 1, dst*6, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     }
     else
```

```
     {
          sprintf(TempBuf, "ТЕН: не працює");
          LCD_WriteString(lcd, 1, dst*6, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);
     }

     switch (settingRAM->mode_heat)
     {
          case  HEATING_OFF:
               sprintf(TempBuf, "Режим 1: ТЕН вимкнено
");
               break;
          case  HEATING_ON:
               sprintf(TempBuf, "Режим 2: ТЕН підтримує T=%d°C в
час.інт", settingRAM->T_max_pool[settingRAM->mode_pump_pool-1] +
settingRAM->dT_min_pump_pool);
               break;
          case  HEATING_ALL:
               sprintf(TempBuf, "Режим 3: ТЕН підтримує T=%d°C
постійно ", settingRAM->T_max_pool[settingRAM->mode_pump_pool-1] +
settingRAM->dT_min_pump_pool);
               break;
     }
     LCD_WriteString(lcd, 1, dst*7, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);

     // VALVE
     if (FLAG_VALVE_CHECK)
     {
          sprintf(TempBuf, "Клапан: відкрито");
          LCD_WriteString(lcd, 1, dst*8, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     }
     else
     {
          sprintf(TempBuf, "Клапан: закрито ");
          LCD_WriteString(lcd, 1, dst*8, utf8_to_win1251(TempBuf,
OutputBuf), &Font_12x20, col_z, col_f, LCD_SYMBOL_PRINT_FAST);
     }

//горизонталні лінії
     LCD_DrawLineH(lcd, 0, dst-2, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, dst*2-2, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, dst*4-2, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, dst*6-2, 319, fm->color_scheme-
>slider_color);
     LCD_DrawLineH(lcd, 0, dst*8-2, 319, fm->color_scheme-
>slider_color);

// дані в таблиці
```

```
    if (packet_RX.t3 < (ERROR_SENSOR-5) ) sprintf(TempBuf, "%.1f",
ave_tPool->Mean(ave_tPool) / 100.0);
    else sprintf(TempBuf, "ERROR");

    LCD_WriteString(lcd, 260, 143, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    sprintf(TempBuf, "%.2f ", ave_V->Mean(ave_V) / 100.0);
    LCD_WriteString(lcd, 260, 163, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    sprintf(TempBuf, "%.2f", ((settingRAM->Q_day / 100.0) /
3600.0));// /100 - переводимо в цілі з сотих, та в кВт год
    LCD_WriteString(lcd, 260, 183, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    if (packet_RX.t1 < (ERROR_SENSOR-5) ) sprintf(TempBuf, "%.1f",
ave_tIn->Mean(ave_tIn) / 100.0);
    else sprintf(TempBuf, "ERROR");

    LCD_WriteString(lcd, 33, 223, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    if (packet_RX.t2 < (ERROR_SENSOR-5) ) sprintf(TempBuf, "%.1f",
ave_tOut->Mean(ave_tOut) / 100.0);
    else sprintf(TempBuf, "ERROR");

    LCD_WriteString(lcd, 110, 223, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    sprintf(TempBuf, "%.1f", packet->dT / 100.0);
    LCD_WriteString(lcd, 185, 223, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);

    sprintf(TempBuf, "%.1f", packet->dT_pool / 100.0);
    LCD_WriteString(lcd, 273, 223, TempBuf, &Font_12x20, col_b,
col_f, LCD_SYMBOL_PRINT_FAST);
}

void printChart3() // Друк графіків на інфо екрані 3, додається одна
точка і графік зсувається
{
    int i;
    int vUp = 125;//відстань між вехніми та нижніми графіками
// написи на графіках
    uint32_t col_b = fm->color_scheme->text_color_file;    //колір
букв
    uint32_t col_f = fm->color_scheme->bg_color;      //колір фону

    LCD_WriteString(lcd, 14, 125, utf8_to_win1251("різниця dT,°C",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, col_f,
LCD_SYMBOL_PRINT_FAST);
```

```
    LCD_WriteString(lcd, 135, 140, "+13", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 135, 160, "+9", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 135, 180, "+5", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 135, 200, "+1", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 135, 220, "-3", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 180, 125, utf8_to_win1251("Витрата, л/хв",
OutputBuf), &Font_12x20, fm->color_scheme->text_color_file, col_f,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 296, 150, "7", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 302, 150, "5", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 296, 180, "5", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 302, 180, "0", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 296, 210, "2", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 302, 210, "5", &Font_8x13, col_b, col_f,
LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 10, 125 - vUp,
utf8_to_win1251("Температура,°C", OutputBuf), &Font_12x20, col_b,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 135, 140 - vUp, "+", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 141, 140 - vUp, "4", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 147, 140 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 135, 160 - vUp, "+", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 141, 160 - vUp, "3", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 147, 160 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 135, 180 - vUp, "+", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 141, 180 - vUp, "2", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 147, 180 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 135, 200 - vUp, "+", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
    LCD_WriteString(lcd, 141, 200 - vUp, "1", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
```

```
     LCD_WriteString(lcd, 147, 200 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 135, 220 - vUp, "+", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 141, 220 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);

     LCD_WriteString(lcd, 180, 125 - vUp, utf8_to_win1251("Енергія,
Дж", OutputBuf), &Font_12x20, col_b, col_f, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 290, 150 - vUp, "1", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 296, 150 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 302, 150 - vUp, "0", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 180 - vUp, "1", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 296, 180 - vUp, "0", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 302, 180 - vUp, "0", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 210 - vUp, "5", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 296, 210 - vUp, "0", &Font_8x13, col_b,
col_f, LCD_SYMBOL_PRINT_PSETBYPSET);

// Горизонтальна шкала по часу
     for(i = 0; i <= 120; i = i + 10)
     {
          LCD_DrawPixel(lcd, 4 + i, 239, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 4 + i, 238, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 167 + i, 239, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 167 + i, 238, COLOR_DARKGREY);

          LCD_DrawPixel(lcd, 4 + i, 239 - vUp, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 4 + i, 238 - vUp, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 167 + i, 239 - vUp, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 167 + i, 238 - vUp, COLOR_DARKGREY);

     }

     int8_t x = 0;
     uint8_t tmp;
     for(i = 0; i < 120; i++)    // Графік зліва на право
     {
     // Вирахувати координати поточної точки x в кільцевому буфері.
Змінюється від 0 до 120-1
          if (chart_dry->posChart < i) x = 120 + chart_dry->posChart
- i; else x = chart_dry->posChart - i;

          LCD_DrawLineV(lcd, 125 - i, 137, 100, fm->color_scheme-
>bg_color);
```

```
          LCD_DrawLineV(lcd, 287 - i, 137, 100, fm->color_scheme-
>bg_color);
          LCD_DrawLineV(lcd, 125 - i, 137 - vUp, 100, fm-
>color_scheme->bg_color);
          LCD_DrawLineV(lcd, 287 - i, 137 - vUp, 100, fm-
>color_scheme->bg_color);

          if (i % 5 == 0) // Пунктирні лінії графіка
          {
               LCD_DrawPixel(lcd, 125 - i, 225, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 205, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 185, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 165, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 145, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 210, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 185, COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 160, COLOR_DARKGREY);

               LCD_DrawPixel(lcd, 125 - i, 225 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 205 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 185 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 165 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 125 - i, 145 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 210 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 185 - vUp,
COLOR_DARKGREY);
               LCD_DrawPixel(lcd, 287 - i, 160 - vUp,
COLOR_DARKGREY);
          }

     // Вивести нову точку

          if ((chart_dry->dTChart[x] == 0) || (chart_dry->dTChart[x]
== 100))
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>dTChart[x], fm->color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>dTChart[x], fm->color_scheme->text_color_file);

          if ((chart_dry->dT_pool_Chart[x] == 0) || (chart_dry-
>dT_pool_Chart[x] == 100))
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>dT_pool_Chart[x], fm->color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>dT_pool_Chart[x], COLOR_RED);
```

```
          if ((chart_dry->V_Chart[x] == 0)  || (chart_dry-
>V_Chart[x] == 100))
               LCD_DrawPixel(lcd, 287 - i, 236 - chart_dry-
>V_Chart[x], fm->color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 287 - i, 236 - chart_dry-
>V_Chart[x], fm->color_scheme->text_color_cursor);


          if ((chart_dry->tPoolChart[x] == 0) || (chart_dry-
>tPoolChart[x] == 100))
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>tPoolChart[x]- vUp, fm->color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 125 - i, 236 - chart_dry-
>tPoolChart[x]- vUp, COLOR_RED);


          tmp = chart_dry->tInChart[x] & 0x7f;   // Відкинути
старший розряд - ознака увімкнення насоса
          if ((tmp == 0) || (tmp == 100))
               LCD_DrawPixel(lcd, 125 - i, 236 - tmp - vUp, fm-
>color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 125 - i, 236 - tmp - vUp, fm-
>color_scheme->text_color_dir);

          tmp = chart_dry->tOutChart[x] & 0x7f;   // Відкинути
старший розряд - ознака увімкнення ТЕН
          if ((tmp ==0) || (tmp == 100))
               LCD_DrawPixel(lcd, 125 - i, 236 - tmp - vUp, fm-
>color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 125 - i, 236 - tmp - vUp, fm-
>color_scheme->text_color_cursor);



          if ((chart_dry->Q_Chart[x] == 0) || (chart_dry->Q_Chart[x]
== 100))
               LCD_DrawPixel(lcd, 287 - i,236 - chart_dry-
>Q_Chart[x] - vUp, fm->color_scheme->text_color_file);
          else
               LCD_DrawPixel(lcd, 287 - i,236 - chart_dry-
>Q_Chart[x] - vUp, fm->color_scheme->text_color_cursor);
     }

     if(FLAG_DEMO_CHECK)
          LCD_WriteString(lcd, 135, 2, utf8_to_win1251("ДЕМО",
OutputBuf), &Font_12x20, COLOR_RED, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);
```

```
}

//   chart_dry->E_Chart[280]; // Отримана енергія
//   chart_dry->posChart_E;   // позиція в масиві графіків - початок
виводу від 0 до 280-1
void printChart_E(void) // Друк графіка отриманої енергії
{
     ScreenMode->Set_Poz(ScreenMode, 37);
     LCD_Fill(lcd, fm->color_scheme->bg_color);
 // Заголовок
    LCD_FillWindow(lcd, 0, 0, 319, 19, fm->color_scheme-
>cursor_color);
     LCD_WriteString(lcd, 17, 3, utf8_to_win1251("Отримана теплова
енергія, кВт·год", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->cursor_color,
LCD_SYMBOL_PRINT_FAST);

     int16_t i, x = 0;
     for(i = 0; i < 280; i++)    // Графік зліва на право
     {
         // Вивести нову точку
        LCD_DrawLineV(lcd, 285 - i, 37 + (200 - chart_dry-
>E_Chart[x]),  chart_dry->E_Chart[x], fm->color_scheme-
>bg_line1_color);
         LCD_DrawPixel(lcd, 285 - i, 236 - chart_dry->E_Chart[x],
fm->color_scheme->text_color_cursor);

// Вирахувати координати поточної точки x в кільцевому буфері.
Змінюється від 0 до 280-1
         if (chart_dry->posChart_E < i) x = 280 + chart_dry-
>posChart_E - i; else x = chart_dry->posChart_E - i;

          if (i % 5 == 0) // Пунктирні лінії графіка
          {
               LCD_DrawPixel(lcd, 285 - i, 195, COLOR_DARKGREY);
//40 - 8
               LCD_DrawPixel(lcd, 285 - i, 155, COLOR_DARKGREY);
//80 - 6
               LCD_DrawPixel(lcd, 285 - i, 115, COLOR_DARKGREY);
//120- 4
               LCD_DrawPixel(lcd, 285 - i, 75, COLOR_DARKGREY);
//160- 2
          }
     }

// написи на графіку
     LCD_WriteString(lcd, 290, 32, "1", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 296, 32, "0", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
```

```
     LCD_WriteString(lcd, 290, 70, "8", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 110, "6", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 150, "4", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 190, "2", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);
     LCD_WriteString(lcd, 290, 227, "0", &Font_8x13, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_PSETBYPSET);

// Горизонтальна шкала по часу
     for(i = 0; i <= 280; i = i + 10)
     {
          LCD_DrawPixel(lcd, 4 + i, 239, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 4 + i, 238, COLOR_DARKGREY);

          LCD_DrawPixel(lcd, 4 + i, 37, COLOR_DARKGREY);
          LCD_DrawPixel(lcd, 4 + i, 38, COLOR_DARKGREY);
     }
}

void get_data_E(void)// заповнення масиву графіка енергії (за
останні 280 днів)
{
   // Малюємо графіки в пам'яті, а коли потрібно то виводимо
     // Працюємо через кільцевий буфер
     // Додати нову точку в кільцевий буфер
     int16_t Q_day_Kw_h = settingRAM->Q_day /3600.0;   //   /3600 -
переводимо з кДж → кВт·год, значення в сотих долях
     settingRAM->Q_day = 0;   //обнуляємо суму отриманої енергії за
добу

//Енергія    0 ... 10 кВт·год, в сотих долях 0 ... 1000, розтягуємо
на 200 точок
     if (Q_day_Kw_h >= 1000)  chart_dry->E_Chart[chart_dry-
>posChart_E] = 199; // Якщо E більше  10 кВт·год  то округляємо до
10
     else chart_dry->E_Chart[chart_dry->posChart_E] = Q_day_Kw_h /
5; // діапазон 0...10

     if (chart_dry->posChart_E < 299) chart_dry->posChart_E++; else
chart_dry->posChart_E = 0; // Змінили положення в буфері і замкнули
буфер
}

void dry_get_data(void)// заповнення масиву графіка
```

```
{
   // Малюємо графіки в пам'яті, а коли потрібно то швидко виводимо
  chart_dry->TimeChart++;
     if ((int16_t)((int16_t)chart_dry-
>TimeChart*TIME_SCAN_SENSOR)>=(int16_t)TIME_PRINT_CHART) //
перевірка чи не пора добавляти точку на графік
     {
          CoolData  =    1;  // Отримано нові дані

     // Працюємо через кільцевий буфер
     // Додати нову точку в кільцевий буфер
// Температура в In, діапазон 0 . . . +50 розтягуємо на 100 точок
          if (ave_tIn->Mean(ave_tIn) <= 0)       chart_dry-
>tInChart[chart_dry->posChart] = 0;       // Якщо температура нижче
0, то округляємо до 0
               else  if (ave_tIn->Mean(ave_tIn) >=5000)  chart_dry-
>tInChart[chart_dry->posChart] = 99;   // Якщо температура вище +50,
то округляємо до 50
                    else chart_dry->tInChart[chart_dry->posChart] =
(ave_tIn->Mean(ave_tIn) ) / 50;   // діапазон 0...+5000 зменшуємо до
100 точок

          if (chart_dry->ChartMotor == 1) chart_dry-
>tInChart[chart_dry->posChart] |= 0x80; // ознака роботи насоса -
старший біт в 1 - колір фона на графіку змінюється
          chart_dry->ChartMotor = 0;

// Температура Out, діапазон 0 . . . +50 розтягуємо на 100 точок
          if (ave_tOut->Mean(ave_tOut) <= 0) chart_dry-
>tOutChart[chart_dry->posChart] = 0;           // Якщо температура
нижче 0, то округляємо до 0
               else  if (ave_tOut->Mean(ave_tOut) >= 5000)
chart_dry->tOutChart[chart_dry->posChart] = 99; // Якщо температура
вище +50, то округляємо до 50
                    else chart_dry->tOutChart[chart_dry->posChart] =
(ave_tOut->Mean(ave_tOut) ) / 50; // діапазон 0...+5000 зменшуємо до
100 точок

          if (chart_dry->ChartHeat == 1) chart_dry-
>tOutChart[chart_dry->posChart] |= 0x80; // ознака роботи нагрівача
- старший біт в 1 - колір фона на графіку змінюється
          chart_dry->ChartHeat = 0;

// Температура Pool, діапазон 0 . . . +50 розтягуємо на 100 точок
          if (ave_tPool->Mean(ave_tPool) <= 0) chart_dry-
>tPoolChart[chart_dry->posChart] = 0;           // Якщо температура
нижче 0, то округляємо до 0
               else  if (ave_tPool->Mean(ave_tPool) >= 5000)
chart_dry->tPoolChart[chart_dry->posChart] = 99; // Якщо температура
вище +50, то округляємо до 50
                    else chart_dry->tPoolChart[chart_dry->posChart]
= (ave_tPool->Mean(ave_tPool) ) / 50; // діапазон 0...+5000
зменшуємо до 100 точок
```

```
// Витрата  0 ... 100 л/хв
          if (ave_V->Mean(ave_V) <= 0) chart_dry->V_Chart[chart_dry-
>posChart] = 0;           // Якщо нижче 0, то округляємо до 0
                else  if (ave_V->Mean(ave_V) >= 10000)  chart_dry-
>V_Chart[chart_dry->posChart] = 99; // Якщо вище 20, то округляємо
до 20
                     else chart_dry->V_Chart[chart_dry->posChart] =
(ave_V->Mean(ave_V) ) / 100; // діапазон 0...+5000 зменшуємо до 100
точок

// Різниця температур - діапазон від -5 до 15 градусів, розтягуємо
на 100 точок
          int16_t temp_dT;
//   temp_dT = ave_tIn->Mean(ave_tIn) - ave_tOut->Mean(ave_tOut);

          temp_dT = packet->dT;
          if (temp_dT <= -500) chart_dry->dTChart[chart_dry-
>posChart] = 0;           // Якщо різниця менше -5 то округляємо до
-5
          else  if (temp_dT >= 1500)  chart_dry->dTChart[chart_dry-
>posChart] = 99; // Якщо різниця більше +15  то округляємо до 15
                else chart_dry->dTChart[chart_dry->posChart] =
(temp_dT + 500) / 20; // діапазон -500...+1500 розтягуємо зменшуємо
до 100 точок

          temp_dT = packet->dT_pool;
          if (temp_dT <= -500) chart_dry->dT_pool_Chart[chart_dry-
>posChart] = 0;           // Якщо різниця менше -5 то округляємо до
-5
          else  if (temp_dT >= 1500)  chart_dry-
>dT_pool_Chart[chart_dry->posChart] = 99; // Якщо різниця більше +15
то округляємо до 15
                else chart_dry->dT_pool_Chart[chart_dry->posChart] =
(temp_dT + 500) / 20; // діапазон -500...+1500 розтягуємо зменшуємо
до 100 точок

//Енергія    0 ... 200 W
          int16_t temp_Q;
          temp_Q = packet->Q_12m;

          if (temp_Q <= 0) chart_dry->Q_Chart[chart_dry->posChart] =
0;           // Якщо Q менше 0 то округляємо до 0
                else  if (temp_Q >= 20000)  chart_dry-
>Q_Chart[chart_dry->posChart] = 99; // Якщо Q більше  200 W  то
округляємо до 200
                     else chart_dry->Q_Chart[chart_dry->posChart] =
temp_Q / 200; // діапазон 0...20000 зменшуємо до 100 точок

          packet->Q_12m = 0; //обнуляємо суму отриманої енергії за
12 хв.
```

```
          if (chart_dry->posChart < 119) chart_dry->posChart++; else
chart_dry->posChart=0;  // Змінили положення в буфері і замкнули
буфер
          chart_dry->TimeChart = 0;  // Зсув графіка і друк нової
точки, скидання лічильника
     }
}

//перевірка чи не пора вимкнути дисплей
void CheckEkran(void)
{
     if (FLAG_EKRAN_CHECK)
     {
          if ((int16_t)(hour_ekran * TIME_SCAN_SENSOR) >=
(int16_t)TIME_EKRAN)
          {
               EKRAN_OFF();
          }
     }
}

//void dry_update(int8_t IsP)
void updateChart3()
{
     if (CoolData == 1) // Якщо отримано нові дані для графіків
     {
         if (chart_dry->TimeChart == 0) //і якщо прийшов час
відображення нової точки
          {
               printChart3(); // то оновлюємо графіки
          }
         CoolData = 0;
     }
}

bool CheckTime_Period()
{
     uint8_t hour_now = sTime.Hours;
     uint8_t min_now = sTime.Minutes;
     if(settingRAM->Use_Period_1 == 1)
     {
          if(settingRAM->PeriodTime_1[0] >= hour_now)
          {
               if(settingRAM->PeriodTime_1[1] >= min_now)
               {
                    if(settingRAM->PeriodTime_1[2] <= hour_now)
                    {
                         if(settingRAM->PeriodTime_1[3] <= min_now)
                         {
                              return true;
                         }
                    }
               }
```

```
            }
        }

        if(settingRAM->Use_Period_2 == 1)
        {
            if(settingRAM->PeriodTime_2[0] >= hour_now)
            {
                if(settingRAM->PeriodTime_2[1] >= min_now)
                {
                    if(settingRAM->PeriodTime_2[2] <= hour_now)
                    {
                        if(settingRAM->PeriodTime_2[3] <= min_now)
                        {
                            return true;
                        }
                    }
                }
            }
        }

        if(settingRAM->Use_Period_3 == 1)
        {
            if(settingRAM->PeriodTime_3[0] >= hour_now)
            {
                if(settingRAM->PeriodTime_3[1] >= min_now)
                {
                    if(settingRAM->PeriodTime_3[2] <= hour_now)
                    {
                        if(settingRAM->PeriodTime_3[3] <= min_now)
                        {
                            return true;
                        }
                    }
                }
            }
        }
        return false;
}

// Перевірка статусу системи
int8_t CheckON(void) //вертає 1, якщо була зміна стану обладнання
{
        if(temp_ready == 0) return 0; // Ігнорувати події до першого
отримання даних з датчиків

        int8_t state = 0;

        //якщо є непрацездатність хоча б одного датчика температури
        if (packet_RX.t1 >= (ERROR_SENSOR-5) || packet_RX.t2 >=
(ERROR_SENSOR-5) || packet_RX.t3 >= (ERROR_SENSOR-5) )
        {
            InsertIvent(11);
```

```
          if (FLAG_HEAT_CHECK) { HeatOFF(); state = 1; }
     // вимкнути ТЕН
          if (FLAG_PUMPPOOL_CHECK) { PumpPoolOFF(); state = 1; }
     //зупинка насоса басейна
          if (FLAG_VALVE_CHECK) { ValveOFF(); state = 1; }
     //закриваємо клапан
          if (FLAG_PUMP_CHECK) { MotorOFF(); state = 1; }        //
вимикаємо насос колектора

          if (packet_RX.t1 >= (ERROR_SENSOR-5))
          {
               switch (ERROR_SENSOR - packet_RX.t1)
               {
                    case  1: // 1 = not present
                         sprintf(ErrorBuf, "Помилка Твх:відсутній
датчик");
                         break;
                    case  2: // 2 = ROM CRC error
                         sprintf(ErrorBuf, "Помилка Твх: ROM CRC
error");
                         break;
                    case  3: // 3 = scratchpad CRC error
                         sprintf(ErrorBuf, "Помилка Твх:scratchpad
CRC error");
                         break;
                    case  4: // 4 = invalid temperature (85 / -127)
                         sprintf(ErrorBuf, "Помилка Твх:invalid
temperature");
                         break;
                    case  5: // 5 = read error
                         sprintf(ErrorBuf, "Помилка Твх: read
error");
                         break;
               }
          }
          else
          if (packet_RX.t2 >= (ERROR_SENSOR-5))
          {
               switch (ERROR_SENSOR - packet_RX.t2)
               {
                    case  1: // 1 = not present
                         sprintf(ErrorBuf, "Помилка Твих:відсутній
датчик");
                         break;
                    case  2: // 2 = ROM CRC error
                         sprintf(ErrorBuf, "Помилка Твих: ROM CRC
error");
                         break;
                    case  3: // 3 = scratchpad CRC error
                         sprintf(ErrorBuf, "Помилка Твих:scratchpad
CRC error");
                         break;
                    case  4: // 4 = invalid temperature (85 / -127)
```

```
                         sprintf(ErrorBuf, "Помилка Твих:invalid
temperature");
                         break;
                    case  5: // 5 = read error
                         sprintf(ErrorBuf, "Помилка Твих: read
error");
                         break;
               }
          }
          else
          if (packet_RX.t3 >= (ERROR_SENSOR-5))
          {
               switch (ERROR_SENSOR - packet_RX.t3)
               {
                    case  1: // 1 = not present
                         sprintf(ErrorBuf, "Помилка Тбас:відсутній
датчик");
                         break;
                    case  2: // 2 = ROM CRC error
                         sprintf(ErrorBuf, "Помилка Тбас: ROM CRC
error");
                         break;
                    case  3: // 3 = scratchpad CRC error
                         sprintf(ErrorBuf, "Помилка Тбас:scratchpad
CRC error");
                         break;
                    case  4: // 4 = invalid temperature (85 / -127)
                         sprintf(ErrorBuf, "Помилка Тбас:invalid
temperature");
                         break;
                    case  5: // 5 = read error
                         sprintf(ErrorBuf, "Помилка Тбас: read
error");
                         break;
               }
          }
          return state;
     }

    // виявлення перегріву колектора
     // якщо температура в колекторі вище максимально-допустимої
(T_max_valve = 70)
    if ( (ave_tIn->Mean(ave_tIn) > (settingRAM->T_max_valve * 100) )
|| ave_tOut->Mean(ave_tOut) > (settingRAM->T_max_valve * 100) )
    {
        InsertIvent(12);

          if (!FLAG_VALVE_CHECK)   //відкриваємо клапан
        {    ValveON(); state = 1; }

        if (!FLAG_PUMP_CHECK) //якщо насос колектора вимкнено
        {    MotorON(); state = 1; }        // то вмикаємо насос
колектора
```

```
        if (!FLAG_PUMPPOOL_CHECK)  //якщо насос басейна вимкнено
        { PumpPoolON(); state = 1; } // запуск насоса басейна

          return state;
    }

//виявлення ситуації коли немає витрати, а насос увімкнено
     if (FLAG_PUMP_CHECK)     //якщо насос колектора працює
     {
         //якщо насос пропрацював не менше 30 сек і немає демо-
режима
          if (( (int16_t)((int16_t)time_motor_ON * TIME_SCAN_SENSOR)
>= 30 ) && !FLAG_DEMO_CHECK )
          {
              if(ave_V->Mean(ave_V) == 0) //якщо витрата дорівнює 0
            {
                InsertIvent(13);
                    sprintf(ErrorBuf, "Помилка: витрата = 0");
                MotorOFF();        // то вимикаємо насос
                    settingRAM->mode_flow = PUMP_KOL_OFF; //змінюємо
на режим "Насос вимкнено"
                    return 1;
               }
        }
    }

//  Контроль заморожування - Вимкнути ТЕН коли температура
підніметься на 0,5 градуса від критичної + значення гістерезису (2
°C)
     if (FLAG_HEAT_CHECK)     //якщо ТЕН працює
     {
          if (settingRAM->mode_heat == HEATING_OFF)    //і режим
нагрівання ТЕНом вимкнено
          {
               if (ave_tOut->Mean(ave_tOut) > (TEMP_MIN +
settingRAM->dT_OFF * 100 + 50) )
               {
                    if (ave_tIn->Mean(ave_tIn) > (TEMP_MIN +
settingRAM->dT_OFF * 100 + 50) )
                    {
                         HeatOFF(); state = 1;
                    }
               }
          }
          else
          {    //якщо Т в басейні вище необхідної (max Т вибраного
режиму)
               if (ave_tPool->Mean(ave_tPool) >= (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100) )
               {
                    HeatOFF(); state = 1;
               }
```

```
          }
     }
     else //if (!FLAG_HEAT_CHECK)  //якщо ТЕН вимкнено
// Контроль заморожування, увімкнення ТЕН
     {
          // якщо температура нижче мінімально допустимої (TEMP_MIN)
          if ( ave_tOut->Mean(ave_tOut) < TEMP_MIN || ave_tIn-
>Mean(ave_tIn) < TEMP_MIN )
          {
               HeatON();// то увімкнення ТЕН
               state = 1;
          }
     }

    //якщо  (Т в басейні нижче необхідної (min Т вибраного режиму))
і  (в теплообміннику Т вище ніж в басейні на dT_min_pump_pool)
    if ( (ave_tPool->Mean(ave_tPool) < (settingRAM-
>T_min_pool[settingRAM->mode_pump_pool-1] * 100) ) &&
               ( packet->dT_pool > settingRAM->dT_min_pump_pool *
100) )
    {//якщо (вибрано режим підтримання Т в басейні)
          if (settingRAM->mode_pump_pool != MODE_6_PUMP_POOL)
          {
               if (!FLAG_PUMPPOOL_CHECK)     //якщо насос басейна
вимкнений
               {
                    //якщо насос відпочив не менше MIN_OFF_TIME, або
ще не запускався, або якщо демо-режим
                    if ( ( (int16_t)((int16_t)time_motor_pool_OFF *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_OFF_TIME * 60 ) ||
time_motor_pool_ON == 0 || FLAG_DEMO_CHECK )
                    {
                         PumpPoolON();// то вмикаємо насос басейна
                         state = 1;
                    }
               }
          }
    }

     if (FLAG_PUMPPOOL_CHECK) //якщо насос басейна працює
     {     // якщо температура в теплообміннику нижче максимально-
допустимої (T_max_valve=70) на більше ніж значення гістерезиса
          if( (ave_tIn->Mean(ave_tIn) < (settingRAM->T_max_valve *
100 - settingRAM->dT_OFF * 100) ) &&  (ave_tOut->Mean(ave_tOut) <
(settingRAM->T_max_valve * 100 - settingRAM->dT_OFF * 100) ))
          {
//якщо клапан був відкритий не менше MIN_ON_TIME або якщо демо-режим

            if (( (int16_t)((int16_t)time_valve_ON *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_ON_TIME  * 60) ||
FLAG_DEMO_CHECK )
               {
                if (FLAG_VALVE_CHECK)
```

```
                {  ValveOFF(); state = 1; }  //закриваємо клапан
            }

//якщо насос пропрацював не менше time_motor_pool_ON або якщо демо-
режим
               if (( (int16_t)((int16_t)time_motor_pool_ON *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_ON_TIME  * 60) ||
FLAG_DEMO_CHECK )
               {
                    if (settingRAM->mode_pump_pool ==
MODE_6_PUMP_POOL)    //якщо режим "Насос вимкнено"
                    {
                    if (!FLAG_VALVE_CHECK) //якщо клапан закритий
                    {
                        PumpPoolOFF();       //зупинка насоса
басейна
                        state = 1;
                    }
                }
                    else //якщо режим циркуляції
                    {//якщо Т в басейні вище необхідної (max Т
вибраного режиму)
                     if ( ave_tPool->Mean(ave_tPool) >
(settingRAM->T_max_pool[settingRAM->mode_pump_pool-1] * 100)  ||
                     ( packet->dT_pool < (settingRAM-
>dT_min_pump_pool - settingRAM->dT_OFF) * 100 ) //або якщо Т в
теплообміннику <= ніж в басейні на (dT-гістерезис)
                     )
                     {    //якщо насос пропрацював не менше
MIN_ON_TIME або якщо демо-режим
                          if ((
(int16_t)((int16_t)time_motor_pool_ON * TIME_SCAN_SENSOR) >=
(int16_t)settingRAM->MIN_ON_TIME  * 60) || FLAG_DEMO_CHECK )
                          {
                            if (!FLAG_VALVE_CHECK) //якщо клапан
закритий
                            {
                                      PumpPoolOFF();       // то
вимикаємо насос
                                      state = 1;
                            }
                              }
                     }
                    }
               }
          }
     }

// Вимкнути ТЕН коли Т стане вище встановленої або коли вийдемо з
часового інтервалу
     if (FLAG_HEAT_CHECK)     //якщо ТЕН увімкнено
     {
// якщо Т в теплообміннику вище встановленої + dT + гістерезис
```

```
          if (
          ( (ave_tOut->Mean(ave_tOut) - (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100)) > (settingRAM-
>dT_min_pump_pool + settingRAM->dT_OFF) * 100 )
          &&
          ( (ave_tIn->Mean(ave_tIn) - (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100) ) > (settingRAM-
>dT_min_pump_pool + settingRAM->dT_OFF) * 100 )
          )
          {
               HeatOFF(); state = 1;    // вимкнення ТЕН
          }
          else
               if(settingRAM->mode_heat == HEATING_ON) //увімкнено
режим нагрівання ТЕНом у часові інтервали
               {
                    if(!CheckTime_Period())  // часовий інтервал не
підходить
                    {
                         HeatOFF(); state = 1; // вимкнення ТЕН
                    }
               }
     }
     else //if (!FLAG_HEAT_CHECK)  //якщо ТЕН вимкнено
// Увімкнути ТЕН коли температура стане нижче встановленої
(T_max_pool[settingRAM->mode_pump_pool-1])
     {
          if (settingRAM->mode_pump_pool != MODE_6_PUMP_POOL)
     //якщо не режим "Насос басейна вимкнено"
          {    // якщо Т в теплообміннику нижче встановленої (min Т
вибраного режиму + dT)
               if (
               ( (ave_tOut->Mean(ave_tOut) - (settingRAM-
>T_min_pool[settingRAM->mode_pump_pool-1] * 100) ) <= settingRAM-
>dT_min_pump_pool  * 100 )
               ||
               ( (ave_tIn->Mean(ave_tIn) - (settingRAM-
>T_min_pool[settingRAM->mode_pump_pool-1] * 100) ) <= settingRAM-
>dT_min_pump_pool  * 100 )
               )
               {
                    //якщо Т в басейні нижче необхідної (max Т
вибраного режиму)
                    if (ave_tPool->Mean(ave_tPool) < (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100) )

                    {
                         //якщо ТЕН відпочив не менше MIN_OFF_TIME,
або ще не вмикався, або якщо демо-режим
                         if ( ( (int16_t)((int16_t)time_heat_OFF *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_OFF_TIME * 60 ) ||
time_heat_ON == 0 || FLAG_DEMO_CHECK )
                         {
```

```
                                if (settingRAM->mode_heat ==
HEATING_ON)    //увімкнено режим нагрівання ТЕНом у часові інтервали
                                {
                                     if(CheckTime_Period())   //
часовий інтервал підходить
                                     {
                                          HeatON(); state = 1;     //
увімкнення ТЕН
                                     }
                                }
                                else if (settingRAM->mode_heat ==
HEATING_ALL)   //увімкнено режим постійного підтримання температури
                                {
                                     HeatON(); state = 1;     //
увімкнення ТЕН
                                }
                           }
                      }
                 }
            }
       }

       if (FLAG_PUMP_CHECK)     //якщо насос колектора працює
       {
            if (settingRAM->mode_flow == PUMP_KOL_OFF)   //якщо режим
"Насос вимкнено"
            {    // якщо Т НЕ вище максимально-допустимої (T_max_valve
= 70)
              if ( (ave_tIn->Mean(ave_tIn) <= (settingRAM->T_max_valve
* 100)) && (ave_tOut->Mean(ave_tOut) <= (settingRAM->T_max_valve *
100)) )
              {
                  MotorOFF();   // то вимикаємо насос колектора
                  state = 1;
              }
          }
            else
            {
                 if ( //якщо режим примусової циркуляції: FLOW_ON або
FLOW_OFF і dT < необхідного значення
                      ( settingRAM->mode_flow == FLOW_OFF &&  (
(packet->dT + settingRAM->dT_OFF * 100) <  (settingRAM->dT_min_pump
* 100) ) ) ||
                       ( settingRAM->mode_flow == FLOW_ON &&  packet-
>dT <=  0 ) ||
                       (ave_tPool->Mean(ave_tPool) >= (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100) ) //якщо  (Т в
басейні вище необхідної (max Т вибраного режиму))
                 )
                 {    //якщо насос пропрацював не менше MIN_ON_TIME
або якщо демо-режим
```

```
                    if (( (int16_t)((int16_t)time_motor_ON *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_ON_TIME  * 60) ||
FLAG_DEMO_CHECK )
                    {
                         MotorOFF();         // то вимикаємо насос
                         state = 1;
                    }
               }
          }
     }
     else //   if (!FLAG_PUMP_CHECK)    //якщо насос колектора
вимкнено
     {//якщо режим примусової циркуляції: FLOW_ON або FLOW_OFF і dT
> необхідного значення
          if (
             ( settingRAM->mode_flow == FLOW_OFF  &&  (packet->dT >=
(settingRAM->dT_min_pump * 100) ) ) ||
             ( settingRAM->mode_flow == FLOW_ON  &&  (packet->dT >
settingRAM->dT_OFF * 100) )// якщо режим FLOW_ON і dT більше за
гістерезис
           )
           {    //якщо насос відпочив не менше MIN_OFF_TIME, або ще
не запускався або якщо демо-режим
               if ( ( (int16_t)((int16_t)time_motor_OFF *
TIME_SCAN_SENSOR) >= (int16_t)settingRAM->MIN_OFF_TIME * 60 ) ||
time_motor_ON == 0 || FLAG_DEMO_CHECK )
               {
               //якщо  (Т в басейні нижче необхідної (max Т
вибраного режиму))
                    if (ave_tPool->Mean(ave_tPool) < (settingRAM-
>T_max_pool[settingRAM->mode_pump_pool-1] * 100) )
                    {
                         MotorON();// то вмикаємо насос колектора
                         state = 1;
                    }
               }
          }
     }
     return state;
}

//друк екрану налаштувань з вибором режиму роботи
void printSettingStatic(void)
{
     ScreenMode->Set_Poz(ScreenMode, 1) ;
     Screens[1]->Set_Poz(Screens[1], 1);

     uint32_t col_b = fm->color_scheme->bg_color;      //колір фону
(back)
     uint32_t col_f = fm->color_scheme->text_color_file;    //колір
букв (front)

     LCD_Fill(lcd, col_b);
```

```
    LCD_FillWindow(lcd, 0, 0, 319, 19, fm->color_scheme-
>cursor_color);
    LCD_WriteString(lcd, 60, 0, utf8_to_win1251("НАЛАШТУВАННЯ",
OutputBuf), &Font_15x25, fm->color_scheme->text_color_cursor, fm-
>color_scheme->cursor_color, LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*2, utf8_to_win1251("Насос
басейна", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 15, dist*3, utf8_to_win1251("Насос
сонячного колектора", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 15, dist*4, utf8_to_win1251("ТЕН",
OutputBuf), &Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*7, utf8_to_win1251("Налаштування
роботи насосів", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*8, utf8_to_win1251("Графік
отриманої енергії", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*9,
utf8_to_win1251("Робочий/Демонстраційний режим", OutputBuf),
&Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*10, utf8_to_win1251("Скидання
значень", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);
    LCD_WriteString(lcd, 15, dist*11, utf8_to_win1251("Час / Дата",
OutputBuf), &Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*12, utf8_to_win1251("Максимально-
допустима температура", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*13, utf8_to_win1251("Колірна
схема", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 15, dist*14, utf8_to_win1251("Точність
ходу годинника", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

//   LCD_WriteString(lcd, 15, dist*15, utf8_to_win1251("----------
----", OutputBuf), &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_FAST);

    LCD_WriteString(lcd, 5, dist*(1+Screens[1]-
>Get_Poz(Screens[1])), ">", &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_PSETBYPSET);
```

```
}

// Друк журнала історії
void printHistoryLog4(void)
{
     LCD_WriteString(lcd, 100, dist*1, utf8_to_win1251("ЖУРНАЛ
ПОДІЙ", OutputBuf), &Font_12x20, fm->color_scheme->text_color_dir,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);

     for(int i=0; i < chart_dry->CountLogs; i++)
     {
          Print_time_date(chart_dry->IventTimeDate[i], 2,
dist*(i+3), fm->color_scheme->text_color_cursor);
          LCD_WriteString(lcd, 130, dist*(i+3),
utf8_to_win1251(Get_Ivent(chart_dry->ID_ivent[i]), OutputBuf),
&Font_12x20, fm->color_scheme->text_color_file, fm->color_scheme-
>bg_color, LCD_SYMBOL_PRINT_FAST);
     }
     show_time();
}

// Оновлення журнала історії
void UpdateHistoryLog4(void)
{
     uint32_t col_b = fm->color_scheme->bg_color;
     LCD_FillWindow(lcd, 0, 36, 319, 239, col_b);

     for(int i=0; i < chart_dry->CountLogs; i++)
     {
          Print_time_date(chart_dry->IventTimeDate[i], 2,
dist*(i+3), fm->color_scheme->text_color_cursor);
          LCD_WriteString(lcd, 130, dist*(i+3),
utf8_to_win1251(Get_Ivent(chart_dry->ID_ivent[i]), OutputBuf),
&Font_12x20, fm->color_scheme->text_color_file, col_b,
LCD_SYMBOL_PRINT_FAST);
     }

     show_time();
}

//отримання опису події
char* Get_Ivent(int8_t ID_IVENT)
{
     char *str;
     switch (ID_IVENT)
     {
          case  0:
               str = "відкриття клапана";
               break;
          case  1:
               str = "закриття клапана";
               break;
          case  2:
```

```
                str = "пуск насоса басейна";
                break;
           case  3:
                str = "стоп насоса басейна";
                break;
           case  4:
                str = "пуск насоса колектора";
                break;
           case  5:
                str = "стоп насоса колектора";
                break;
           case  6:
                str = "увімкнення ТЕН";
                break;
           case  7:
                str = "вимкнення ТЕН";
                break;
           case  8:
                str = "запуск системи";
                break;
           case  9:
                str = "демо-режим увімкнено";
                break;
           case  10:
                str = "демо-режим вимкнено";
                break;
           case  11:
                str = "відмова датчиків Т";
                break;
           case  12:
                str = "перегів колектора";
                break;
           case  13:
                str = "помилка: витрата = 0";
                break;
      }

      char *mem = (char *)calloc(my_strlen(str) + 1, 1);
      if (mem) my_strcpy(mem, str);
      return mem;
}


// Друк значень на екрані 0
void printInfoUpdate0(void)
{
#ifdef  USE_ADC
//Температура МК
      sprintf(TempBuf, "%.1f", ave_t_MK->Mean(ave_t_MK) / 100.0);
      LCD_WriteString(lcd, 210, dist*1, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
//Напруга на МК
```

```
     sprintf(TempBuf, "%.1f", ave_V_MK->Mean(ave_V_MK) / 100.0);
     LCD_WriteString(lcd, 210, dist*2, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
#endif
//Час роботи системи:
     Print_period(settingRAM->hour_unit, 210, dist*5, fm-
>color_scheme->text_color_file);
//Час роботи насоса басейна:
     Print_period(settingRAM->hour_motor_pool, 210, dist*6, fm-
>color_scheme->text_color_file);
//Час роботи насоса колектора:
     Print_period(settingRAM->hour_motor, 210, dist*7, fm-
>color_scheme->text_color_file);
//Час роботи ТЕН:
     Print_period(settingRAM->hour_heat, 210, dist*8, fm-
>color_scheme->text_color_file);

//Останній час роботи
     LCD_WriteString(lcd, 210, dist*11, my_strsplice (2, " / ",
Get_period(time_motor_pool_ON), Get_period(time_motor_pool_OFF)),
&Font_12x20, fm->color_scheme->text_color_file, fm->color_scheme-
>bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 210, dist*12, my_strsplice (2, " / ",
Get_period(time_motor_ON), Get_period(time_motor_OFF)), &Font_12x20,
fm->color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 210, dist*13, my_strsplice (2, " / ",
Get_period(time_heat_ON), Get_period(time_heat_OFF)), &Font_12x20,
fm->color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

     show_time();
}

// Друк статичної картинки екрана 1
void printInfoStatic1(void)
{
     LCD_WriteString(lcd, 2, dist*1, utf8_to_win1251("1. Гістерезис
dT, °C:", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*2, utf8_to_win1251("2. dT
колектора, °C:", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*3, utf8_to_win1251("3. dT басейна,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_file,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*4, utf8_to_win1251("4.
Макс.допустима T,°C:", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
```

```
     LCD_WriteString(lcd, 2, dist*6, utf8_to_win1251("5. Max T бас,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*7, utf8_to_win1251("6. Min T бас,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);

     LCD_WriteString(lcd, 2, dist*8, utf8_to_win1251("7. Max T вх,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*9, utf8_to_win1251("8. Max T вих,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);

     LCD_WriteString(lcd, 2, dist*10, utf8_to_win1251("9. Min T вх,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);
     LCD_WriteString(lcd, 2, dist*11, utf8_to_win1251("10.Min T вих,
°C:", OutputBuf), &Font_12x20, fm->color_scheme->text_color_cursor,
fm->color_scheme->bg_color, LCD_SYMBOL_PRINT_FAST);

     LCD_WriteString(lcd, 2, dist*13, utf8_to_win1251("11.Останнє
збереження:", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

     LCD_WriteString(lcd, 2, dist*14, utf8_to_win1251("12.Помилки
радіобміну:", OutputBuf), &Font_12x20, fm->color_scheme-
>text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
}

// Друк значень на екрані 1
void printInfoUpdate1(void)
{
//   1. Гістерезис dT:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->dT_OFF));
     LCD_WriteString(lcd, 200, dist*1, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

//   2. Значення dT насоса колектора, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->dT_min_pump));
     LCD_WriteString(lcd, 200, dist*2, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

//   3. Значення dT насоса басейна, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM-
>dT_min_pump_pool));
```

```
     LCD_WriteString(lcd, 200, dist*3, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

//   4. Значення T для запобіжного режиму, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->T_max_valve));
     LCD_WriteString(lcd, 200, dist*4, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

//7. Max T бас, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tPoolMax/100.0));
     LCD_WriteString(lcd, 150, dist*6, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tPoolMaxTimeDate, 200, dist*6, fm-
>color_scheme->text_color_cursor);

//8. Max T бас, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tPoolMin/100.0));
     LCD_WriteString(lcd, 150, dist*7, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tPoolMinTimeDate, 200, dist*7, fm-
>color_scheme->text_color_cursor);

//7. Max T вх, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tInMax/100.0));
     LCD_WriteString(lcd, 150, dist*8, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tInMaxTimeDate, 200, dist*8, fm-
>color_scheme->text_color_cursor);

//8. Max T вих, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tOutMax/100.0));
     LCD_WriteString(lcd, 150, dist*9, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tOutMaxTimeDate, 200, dist*9, fm-
>color_scheme->text_color_cursor);

//9. Min T вх, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tInMin/100.0));
     LCD_WriteString(lcd, 150, dist*10, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tInMinTimeDate, 200, dist*10, fm-
>color_scheme->text_color_cursor);

//10. Min T вих, °C:
     sprintf(TempBuf, "%.1f", (float)(settingRAM->tOutMin/100.0));
```

```
     LCD_WriteString(lcd, 150, dist*11, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_cursor, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);
     Print_time_date(settingRAM->tOutMinTimeDate, 200, dist*11, fm-
>color_scheme->text_color_cursor);

//11. Останнє збереження:
     Print_time_date(settingRAM->SaveTimeDate, 200, dist*13, fm-
>color_scheme->text_color_file);

//10. Помилки при радіопередачі
     sprintf(TempBuf, "TX= %d, RX= %d\n", error_TX, error_RX-1);
     LCD_WriteString(lcd, 200, dist*14, TempBuf, &Font_12x20, fm-
>color_scheme->text_color_file, fm->color_scheme->bg_color,
LCD_SYMBOL_PRINT_FAST);

     show_time();
}

//додавання ID події в журнал історії
void InsertIvent(int8_t  ID_IVENT)
{
     for (int8_t k = chart_dry->CountLogs; k > 0; k--)
     {
          if(k < 12)
          {
               chart_dry->IventTimeDate[k][0] = chart_dry-
>IventTimeDate[k-1][0];
               chart_dry->IventTimeDate[k][1] = chart_dry-
>IventTimeDate[k-1][1];
               chart_dry->IventTimeDate[k][2] = chart_dry-
>IventTimeDate[k-1][2];
               chart_dry->IventTimeDate[k][3] = chart_dry-
>IventTimeDate[k-1][3];
               chart_dry->IventTimeDate[k][4] = chart_dry-
>IventTimeDate[k-1][4];
               chart_dry->ID_ivent[k] = chart_dry->ID_ivent[k-1];
          }
     }

     chart_dry->IventTimeDate[0][0] = sTime.Minutes;
     chart_dry->IventTimeDate[0][1] = sTime.Hours;
     chart_dry->IventTimeDate[0][2] = sDate.Date;
     chart_dry->IventTimeDate[0][3] = sDate.Month;
     chart_dry->IventTimeDate[0][4] = sDate.Year;
     chart_dry->ID_ivent[0] = ID_IVENT;

     if(chart_dry->CountLogs < 12)
     {
          chart_dry->CountLogs++;
     }
}
```

```
void ValveON(void)  //відкриття клапана
{
     VALVE_ON();
     time_valve_ON = 0;
     InsertIvent(0);
}

void ValveOFF(void) //закриття клапана
{
     VALVE_OFF();
     time_valve_OFF = 0;
     InsertIvent(1);
}

void PumpPoolON(void)    //увімкнення насоса
{
     PUMPPOOL_ON();   //увімкнення живлення насоса
     time_motor_pool_ON = 0;
     InsertIvent(2);
}

void PumpPoolOFF(void)   //вимкнення насоса
{
     PUMPPOOL_OFF();     //вимкнення живлення насоса
     time_motor_pool_OFF = 0;
     InsertIvent(3);
}

void MotorON(void)  //увімкнення насоса
{
     MOTOR_ON();       //увімкнення живлення насоса
     time_motor_ON = 0;
     InsertIvent(4);
}

void MotorOFF(void) //вимкнення насоса
{
     MOTOR_OFF();   //вимкнення живлення насоса
     time_motor_OFF = 0;
     InsertIvent(5);
}

void HeatON(void)   //увімкнення ТЕН
{
     HEAT_ON();        //увімкнення живлення ТЕН
     time_heat_ON = 0;
     InsertIvent(6);
}

void HeatOFF(void)  //вимкнення ТЕН
{
     HEAT_OFF();    //вимкнення живлення ТЕН
     time_heat_OFF = 0;
```

```
     InsertIvent(7);
}

void readNRF24(void) //отримання даних з радіомодуля NRF24L01
{
     irq_RX = 0;
     if(nrf24_data_available())
     {
         uint8_t status = nrf24_r_status();

         if (status & (1 << RX_DR))  // Дані отримані успішно
         {
               nrf24_receive(dataRX, PLD_SIZE);
               BufferToPacket_RX(dataRX, &packet_RX);
            error_RX = 0;          //скидаємо кількість невдалих
отримань
               temp_ready = 1;          //ознака, що отримано дані з
датчиків

#if DEBUG
               printf("T1: %d (%.2f C)\n", packet_RX.t1,
packet_RX.t1 / 100.0);
               printf("T2: %d (%.2f C)\n", packet_RX.t2,
packet_RX.t2 / 100.0);
               printf("T3: %d (%.2f C)\n", packet_RX.t3,
packet_RX.t3 / 100.0);
               printf("V: %d (%.2f l/min)\n", packet_RX.V,
packet_RX.V / 100.0);
#endif

               // Отримуємо з датчиків значення температур в сотих
градуса °C
               /* 1-й DS18B20 */
               if (packet_RX.t1 < (ERROR_SENSOR-5)) // перевірка
працездатності датчика температури
               {
                    packet->tIn = packet_RX.t1;
                    ave_tIn->Push(ave_tIn, packet->tIn);
               }

               /* 2-й DS18B20 */
               if (packet_RX.t2 < (ERROR_SENSOR-5))
               {
                    packet->tOut = packet_RX.t2;
                    ave_tOut->Push(ave_tOut, packet->tOut);
               }

               packet->dT = ave_tIn->Mean(ave_tIn) - ave_tOut-
>Mean(ave_tOut);

               /* 3-й DS18B20 */
               if (packet_RX.t3 < (ERROR_SENSOR-5))
               {
```

```
                    packet->tPool = packet_RX.t3;
                    ave_tPool->Push(ave_tPool, packet->tPool);
                }

                packet->dT_pool = ave_tIn->Mean(ave_tIn) - ave_tPool-
>Mean(ave_tPool);

                int16_t V_flow = packet_RX.V; // Отримуємо значення
витрати
                packet->V = V_flow;
                ave_V->Push(ave_V, packet->V);


// ΔQ=c·ρ·V·ΔT·Δt - кількість отриманої енергії від сонячного колектора
за час Δt
// ΔQ — кількість отриманої теплової енергії (в Джоулях) або
кВт·год: QкВт = QДж / 3 600 000
// c — питома теплоємність води (≈ 4186 Дж/(кг·°C))
// ρ — густина води (≈ 1000 кг/м³)
// V — об’ємна витрата води (м³/с)
// ΔT — різниця температур (°C)
// Δt — час протікання (с)

// packet->Q = packet->Q + 4.186 * ( (V_flow/100) / 60) * (packet-
>dT / 100) * 3;
                packet->Q_3s = 4.186f * (V_flow/6000.0f) * (packet-
>dT/100.0f) * 3.0f * 100.0f;

                packet->Q_12m += packet->Q_3s;
                settingRAM->Q_day += packet->Q_3s;

                packet_TX.flags = settingRAM->flags;
            // Перетворюємо в байти і відправляємо
                packetToByteArray_TX(&packet_TX, dataTX);

            //після успішного отримання передаємо дані
                nrf24_stop_listen();
                HAL_Delay(300);
                nrf24_transmit(dataTX, PLD_SIZE);
                HAL_Delay(100);

                nrf24_clear_tx_ds();
                nrf24_listen();
        }
        else
        { // Помилка при отриманні
#if DEBUG
            printf("ERROR RX!\n");
#endif
            nrf24_clear_rx_dr(); // Очистити прапорець
               nrf24_flush_rx(); // Очистити буфер

               nrf24_stop_listen();
```

```
                HAL_Delay(300);
                nrf24_listen();
            }
        }
}

  //обробник переривань таймера (раз на 3 секунди)
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
#if  !DEBUG
     HAL_IWDG_Refresh(&hiwdg); // Скидання сторожового таймера
#endif

     if(htim->Instance == TIM1) //перевіряємо чи переривання від 1-
го таймера TIM1
     {
         irq_3sec = 1;
         error_RX++;
     }
}

void ToDo_every_3sec(void)    //виконуємо кожні 3 секунди
{
#ifdef  USE_ADC
     if(ScreenMode->Get_Poz(ScreenMode) == 0)// якщо знаходимся на
якомусь інфо екрані
     {
          float Vdda = 3.3;        // Напруга живлення (припустимо
3.3 В)
          float V25 = 0.76;        // Напруга при 25°C (з datasheet)
          float Avg_Slope = 0.0025; // Середній нахил, В/°C (з
datasheet)

     // Обчислення Vdda - напруга на МК
          uint32_t vrefintValue = Read_ADC(&hadc1,
ADC_CHANNEL_VREFINT);

          uint16_t VREFINT_CAL = *(__IO uint16_t *)0x1FFF7A2A; //
Калібрувальне значення з пам’яті
          float VREFINT_CAL_VOLT = 3.3;                        //
Напруга калібрування (з datasheet)
          Vdda = (VREFINT_CAL * VREFINT_CAL_VOLT) / vrefintValue;

     // Обчислення температури МК
          uint32_t temp_raw = Read_ADC(&hadc1,
ADC_CHANNEL_TEMPSENSOR);
          float voltage = (temp_raw * Vdda) / 4095.0;      //
Переведення в напругу (12-бітний АЦП)
          float temperature = ((voltage - V25) / Avg_Slope) + 25.0;
// Формула з datasheet

          ave_t_MK->Push(ave_t_MK, temperature*100);
          ave_V_MK->Push(ave_V_MK, Vdda*100);
```

```
     }
#endif

     CheckEkran();       //перевірка чи не пора вимкнути дисплей
     get_time(0);        // отримання актуальних дати, часу і
формування масиву TimeBuf для виводу

     //якщо 30 сек не було отримано даних по радіоканалу
     if(error_RX == 10)//то перезавантажуємо NRF24
     {
          nrf24_stop_listen();
     }
     if(error_RX > 10)
     {
          nrf24_listen();
          error_RX = 0;
     }

     int8_t  state = CheckON();    // Перевірка стану системи
     if (state == 1) { writeSPIFlash(); } //Якщо була зміна стану
пристрою, то запис на flash


     if (FLAG_PUMP_CHECK)     //якщо насос працює
     {
          chart_dry->ChartMotor = 1;  //ознака того що потрібно
показувати увімкнення насоса на графіках
          settingRAM->hour_motor++;   //облік всього часу роботи
насоса
          time_motor_ON++;            //облік  часу роботи насоса
останнього включення
     }
     else
     {
          time_motor_OFF++;   //облік часу зупину насоса
     }

     if (FLAG_HEAT_CHECK)     //якщо ТЕН працює
     {
          chart_dry->ChartHeat = 1;//ознака того що потрібно
показувати увімкнення ТЕН на графіках
          settingRAM->hour_heat++;//облік часу роботи ТЕН
          time_heat_ON++;
     }
     else
     {
          time_heat_OFF++;    //облік часу вимкнення ТЕН
     }

     if (FLAG_PUMPPOOL_CHECK) //якщо насос басейна працює
     {
          settingRAM->hour_motor_pool++;   //облік всього часу
роботи насоса басейна
```

```
          time_motor_pool_ON++; //облік  часу роботи насоса
останнього включення
     }
     else
     {
          time_motor_pool_OFF++;   //облік часу зупину насоса
    }

     if (FLAG_VALVE_CHECK)    //якщо насос басейна працює
     {
          settingRAM->hour_motor_pool++;   //облік всього часу
роботи насоса басейна
          time_valve_ON++; //облік  часу останнього відкритого
клапана
     }
     else
     {
          time_valve_OFF++;   //облік часу закритого клапана
    }

     if (settingRAM->tInMax < ave_tIn->Mean(ave_tIn))// знаходження
максимальної вхідної температури
     {
          settingRAM->tInMax = ave_tIn->Mean(ave_tIn);
          settingRAM->tInMaxTimeDate[0] = sTime.Minutes;
          settingRAM->tInMaxTimeDate[1] = sTime.Hours;
          settingRAM->tInMaxTimeDate[2] = sDate.Date;
          settingRAM->tInMaxTimeDate[3] = sDate.Month;
          settingRAM->tInMaxTimeDate[4] = sDate.Year;
     }
     else
     {
          if (settingRAM->tInMin > ave_tIn->Mean(ave_tIn))//
знаходження мінімальної вхідної температури
          {
               settingRAM->tInMin = ave_tIn->Mean(ave_tIn);
               settingRAM->tInMinTimeDate[0] = sTime.Minutes;
               settingRAM->tInMinTimeDate[1] = sTime.Hours;
               settingRAM->tInMinTimeDate[2] = sDate.Date;
               settingRAM->tInMinTimeDate[3] = sDate.Month;
               settingRAM->tInMinTimeDate[4] = sDate.Year;
          }
     }

     if (settingRAM->tOutMax < ave_tOut->Mean(ave_tOut))    //
знаходження максимальної вихідої температури
     {
          settingRAM->tOutMax = ave_tOut->Mean(ave_tOut);
          settingRAM->tOutMaxTimeDate[0] = sTime.Minutes;
          settingRAM->tOutMaxTimeDate[1] = sTime.Hours;
          settingRAM->tOutMaxTimeDate[2] = sDate.Date;
          settingRAM->tOutMaxTimeDate[3] = sDate.Month;
          settingRAM->tOutMaxTimeDate[4] = sDate.Year;
```

```
     }
     else
     {
          if (settingRAM->tOutMin > ave_tOut->Mean(ave_tOut)) //
знаходження мінімальної вихідої температури
          {
               settingRAM->tOutMin = ave_tOut->Mean(ave_tOut);
               settingRAM->tOutMinTimeDate[0] = sTime.Minutes;
               settingRAM->tOutMinTimeDate[1] = sTime.Hours;
               settingRAM->tOutMinTimeDate[2] = sDate.Date;
               settingRAM->tOutMinTimeDate[3] = sDate.Month;
               settingRAM->tOutMinTimeDate[4] = sDate.Year;
          }
     }

     if (settingRAM->tPoolMax < ave_tPool->Mean(ave_tPool)) //
знаходження максимальної температури басейна
     {
          settingRAM->tPoolMax = ave_tPool->Mean(ave_tPool);
          settingRAM->tPoolMaxTimeDate[0] = sTime.Minutes;
          settingRAM->tPoolMaxTimeDate[1] = sTime.Hours;
          settingRAM->tPoolMaxTimeDate[2] = sDate.Date;
          settingRAM->tPoolMaxTimeDate[3] = sDate.Month;
          settingRAM->tPoolMaxTimeDate[4] = sDate.Year;
     }
     else
     {
          if (settingRAM->tPoolMin > ave_tPool->Mean(ave_tPool)) //
знаходження мінімальної температури басейна
          {
               settingRAM->tPoolMin = ave_tPool->Mean(ave_tPool);
               settingRAM->tPoolMinTimeDate[0] = sTime.Minutes;
               settingRAM->tPoolMinTimeDate[1] = sTime.Hours;
               settingRAM->tPoolMinTimeDate[2] = sDate.Date;
               settingRAM->tPoolMinTimeDate[3] = sDate.Month;
               settingRAM->tPoolMinTimeDate[4] = sDate.Year;
          }
     }

     settingRAM->hour_unit++; //облік часу роботи системи

//   Оновлюємо максимум і мінімум температур на flash
//   Дані оновлюються раз в 3 секунды (TIME_SCAN_SENSOR),
//   а пишемо на flash раз в 3 години для економії ресурсу flash,
кількість циклів перезапису обмежена
     tick_SPI_Flash++;

     if (!FLAG_DEMO_CHECK)//якщо немає демо-режима
     {
          if (((int16_t)tick_SPI_Flash) >= (int16_t)TIME_HOUR) // чи
пора писати на flash
          {
               writeSPIFlash();// Запис на flash
```

```
          }
     }
     if (FLAG_EKRAN_CHECK)
     {
          hour_ekran++;  //облік часу увімкненого дисплея
     }
     dry_get_data();     // заповнення масиву для графіків
     packet_ready = 1;   //встановлюємо ознаку «дані оброблено»
}
// обробник натискання кнопок
// функція вертає 1, якщо змінився екран і потрібно надрукувати
новий
int8_t button_click_handler(void)
{
     hour_ekran     =    0;   //скидання часу роботи дисплея після
останнього натискання кнопки
     uint32_t keys = 0;
     keys = KEYB_Inkeys();         //зчитуємо стан кнопок з буфера
     if (!FLAG_EKRAN_CHECK)        //якщо дисплей потушений
     {
          EKRAN_ON();         //то вмикаємо дисплей
          return 1;      //вертаємо 1 щоб оновити поточний екран,
так як при потушеному екрані дані та графіки не передруковувались
     }
     uint32_t col_b = fm->color_scheme->bg_color;      //колір фону
(back)
     uint32_t col_f = fm->color_scheme->text_color_file;    //колір
букв (front)
     uint8_t poz_S;
     uint8_t poz_SM;
     poz_SM = ScreenMode->Get_Poz(ScreenMode);

// Обробка натискання кнопок
//вхід/вихід(без збереження змін) в/з меню налаштувань
     if ( keys & (1u<<KEYB_RIGHT)  )    //якщо нажали KEYB_RIGHT
(MENU)
     {
          if(poz_SM == 0)     //і якщо знаходимся на інфо екрані
          {
               printSettingStatic();    //то переходимо на екран
налаштувань з вибором режиму роботи
               return 0;
          }
          else //якщо знаходимся не на інфо екрані
          {
               ScreenMode->Set_Poz(ScreenMode, 0);     //то виходимо
з поточного екрана без збереження змін на інфо екран
               return 1;                                // змінився
екран - потрібно надрукувати новий
          }
     }
```

```
     if ( (keys & (1u<<KEYB_UP)) || (keys & (1u<<KEYB_DOWN)) )
     //якщо нажали KEYB_UP чи KEYB_DOWN
     {
          switch (poz_SM)     //стираємо символ ">" (друкуємо
кольором фона на старому місці)
          {
               case 1: //якщо знаходимся на екрані налаштувань
                    LCD_WriteString(lcd, 5, dist * (1 + Screens[1]-
>Get_Poz(Screens[1])), ">", &Font_12x20, col_b, col_b,
LCD_SYMBOL_PRINT_PSETBYPSET);
                    break;
               case 9: //якщо знаходимся на екрані "Колірна схема"
               case 10: //якщо знаходимся на екрані "Демонстраційний
режим"
               case 11: //якщо знаходимся на екрані "Налаштування
ТЕН"
               case 14: //якщо знаходимся на екрані "ТЕН"
               case 38: //якщо знаходимся на екрані "Вибір режиму
роботи насоса басейна"
                    LCD_WriteString(lcd, 5, dist * (1 +
Screens[poz_SM]->Get_Poz(Screens[poz_SM])), ">", &Font_12x20, col_b,
col_b, LCD_SYMBOL_PRINT_PSETBYPSET);
                    break;
               case 37: //якщо знаходимся на екрані графіка енергії
                    ScreenMode->Set_Poz(ScreenMode, 0);
                    return 1;
          }

          if  (keys & (1u<<KEYB_UP)) //якщо нажали KEYB_UP
          {
               switch (poz_SM)     //перехід до іншого екрану вперед
(чи до іншої позиції на екрані налаштувань)
               {
                    case 1: //якщо знаходимся на екрані налаштувань
                         if(Screens[1]->Get_Poz(Screens[1]) == 6) {
Screens[poz_SM]->Down(Screens[poz_SM]); Screens[poz_SM]-
>Down(Screens[poz_SM]);}
                         Screens[poz_SM]->Down(Screens[poz_SM]);
     //перехід
                         break;
                    case 9: //якщо знаходимся на екрані "Колірна
схема"
                    case 11: //якщо знаходимся на екрані
"Налаштування ТЕН"
                    case 14: //якщо знаходимся на екрані "ТЕН"
                    case 38: //якщо знаходимся на екрані "Вибір
режиму роботи насоса басейна"
                         Screens[poz_SM]->Down(Screens[poz_SM]);
//перехід
                         break;
                    default:
                         Screens[poz_SM]->Up(Screens[poz_SM]);
     //перехід
```

```
                }
            }
            else //якщо нажали KEYB_DOWN
            {
                switch (poz_SM)     //перехід до іншого екрану вперед
(чи до іншої позиції на екрані налаштувань)
                {
                    case 1: //якщо знаходимся на екрані налаштувань
                        if(Screens[1]->Get_Poz(Screens[1]) == 3) {
Screens[poz_SM]->Up(Screens[poz_SM]); Screens[poz_SM]-
>Up(Screens[poz_SM]);}
                        Screens[poz_SM]->Up(Screens[poz_SM]);
     //перехід до іншого екрану вперед (чи до іншої позиції на
екрані налаштувань)
                        break;
                    case 9: //якщо знаходимся на екрані "Колірна
схема"
                    case 11: //якщо знаходимся на екрані
"Налаштування ТЕН"
                    case 14: //якщо знаходимся на екрані "ТЕН"
                    case 38: //якщо знаходимся на екрані "Вибір
режиму роботи насоса басейна"
                        Screens[poz_SM]->Up(Screens[poz_SM]);
//перехід
                        break;
                    default:
                        Screens[poz_SM]->Down(Screens[poz_SM]);
     //перехід
                }
            }

            switch (poz_SM)     //друкуємо символ ">" або оновлюємо
екран
            {
                case 1: //якщо знаходимся на екрані налаштувань
                    //друкуємо символ ">" в новому місці
                    LCD_WriteString(lcd, 5, dist * (1 + Screens[1]-
>Get_Poz(Screens[1])), ">", &Font_12x20, col_f, col_b,
LCD_SYMBOL_PRINT_PSETBYPSET);
                    return 0;
                case 9:  //якщо знаходимся на екрані "Колірна схема"
                case 10: //якщо знаходимся на екрані "Демонстраційний
режим"
                case 38: //якщо знаходимся на екрані "Вибір режиму
роботи насоса басейна"
                    if(Screens[poz_SM]->Get_Poz(Screens[poz_SM]) ==
7 )  //пропускаєм рядок
                    {
                        if (keys & (1u<<KEYB_UP))     //якщо нажали
KEYB_UP
                            Screens[poz_SM]-
>Down(Screens[poz_SM]);
                        else //якщо нажали KEYB_DOWN
```

```
                              Screens[poz_SM]->Up(Screens[poz_SM]);
                    }
                    LCD_WriteString(lcd, 5, dist * (1 +
Screens[poz_SM]->Get_Poz(Screens[poz_SM])), ">", &Font_12x20, col_f,
col_b, LCD_SYMBOL_PRINT_PSETBYPSET);
                    return 0;
               case 12: // якщо на екрані "Коригування RTC"
                    printCorRTC_Update();    // Оновлення екрану
"Коригування RTC"
                    return 0;
               case 13: // якщо на екрані "Коригування T для
запобіжного режиму"
                    print_T_valve_Update();  // Оновлення екрану
"Коригування макс.T для запобіжного режиму"
                    return 0;
               case 11: //якщо знаходимся на екрані "Налаштування
ТЕН"
               case 14: //якщо знаходимся на екрані "ТЕН"
                    if(Screens[poz_SM]->Get_Poz(Screens[poz_SM]) ==
4 )  //пропускаєм рядок
                    {
                         if (keys & (1u<<KEYB_UP))     //якщо нажали
KEYB_UP
                              Screens[poz_SM]-
>Down(Screens[poz_SM]);
                         else //якщо нажали KEYB_DOWN
                              Screens[poz_SM]->Up(Screens[poz_SM]);
                    }
                    LCD_WriteString(lcd, 5, dist * (1 +
Screens[poz_SM]->Get_Poz(Screens[poz_SM])), ">", &Font_12x20, col_f,
col_b, LCD_SYMBOL_PRINT_PSETBYPSET);
                    return 0;
               case 32: // якщо на екрані "Коригування dT"
                    printCor_dT_Update();    // Оновлення екрану
"Коригування dT колектора"
                    return 0;
               case 50: // якщо на екрані "Коригування dT"
                    printCor_dT_pool_Update();    // Оновлення
екрану "Коригування dT басейна"
                    return 0;

               default:
                    if (poz_SM >= 2 && poz_SM <= 8)    // якщо на
екрані "Налаштування Часу/Дати"
                    {
                         printSetTimeUpdate();    // Оновлення
екрану "Налаштування Часу/Дати"
                         return 0;
                    }

                    if (poz_SM >= 16 && poz_SM <= 31)  // якщо на
екрані "Налаштування Часу ТЕН"
                    {
```

```
                         printCor_Time_Update();  // Оновлення
екрану "Налаштування Часу ТЕН"
                         return 0;
                    }

                    if (poz_SM >= 33 && poz_SM <= 36)  // якщо на
екрані "Робота насоса"
                    {
                         print_Work_Pump_Update();     // Оновлення
екрану
                         return 0;
                    }

                    if (poz_SM >= 39 && poz_SM <= 49)  // якщо на
екрані "Робота насоса"
                    {
                         print_Modes_T_Pool_Update();  // Оновлення
екрану
                         return 0;
                    }
          }
          return 1;                                   // змінився екран
- потрібно надрукувати новий
     }

     if ( (keys & (1u<<KEYB_LEFT)) )    //якщо нажали KEYB_LEFT (OK)
     {
          if(poz_SM == 0)     // якщо знаходимся на якомусь інфо
екрані
          {
               if (FLAG_EKRAN_CHECK) EKRAN_OFF(); //тушим дисплей
               return 0;
          }

          if (poz_SM == 1)               // якщо знаходимся на екрані
налаштувань
          {
               // обробник нажимання в залежності від положення
курсора
               poz_S = Screens[1]->Get_Poz(Screens[1]);
               switch (poz_S)
               {
                    case 1:
                         print_Mode_PUMP_POOL();  // "Вибір режиму
роботи насоса басейна"
                         break;
                    case 2:
                         print_Mode_PUMP_KOL();   // "Вибір режиму
роботи насоса сонячного колектора"
                         break;
                    case 3: // Налаштування ТЕН
                         printOptionTEN(); // Екран "Налаштування
ТЕН"
```

```
                    break;


                case 6: // Робота насоса
                    print_Work_Pump();  // Друк екрану "Робота
насоса"
                    print_Work_Pump_Update();     // Оновлення
екрану "Робота насоса"
                    break;
                case 7: // Графік отриманої енергії
                    printChart_E();
                    break;
                case 8: // Вимкнено / Увімкнено демонстраційний
режим
                    printDemoMode();
                    break;
                case 9: // Скидання min max температур
                    resetTemp();
                    break;
                case 10: // Друк екрану "Налаштування Часу/Дати
                    printSetTimeStatic();
                    printSetTimeUpdate();
                    break;
                case 11: // Друк екрану "Коригування T для
запобіжного режиму"
                    print_T_valve();
                    print_T_valve_Update();
                    break;
                case 12: // Друк екрану "Колірна схема"
                    printColorMode();
                    break;
                case 13: // Коригування RTC
                    printCorRTC(); // Друк екрану "Коригування
RTC"
                    printCorRTC_Update();    // Оновлення
екрану "Коригування RTC"
                    break;
            }

            if(poz_S == 9) //якщо було вибрано "Скидання min max
температур"
            {
                ScreenMode->Set_Poz(ScreenMode, 0);
     //то встановлюємо поточним екраном - інфо екран
                return 1;                                   //
змінився екран - потрібно надрукувати новий
            }
            else
            { return 0; }
        }

        if (poz_SM < 8)     //якщо на екрані налаштування
Часу/Дати
```

```
          {
               if(poz_SM == 7)
               {
                    if(Screens[7]->Get_Poz(Screens[7]) == 0)
     //якщо нажали Так
                    {
                         SetTime();
                    }
                    ScreenMode->Set_Poz(ScreenMode, 0);
                    return 1;                                   //
змінився екран - потрібно надрукувати новий
               }
               else
               {
                    if(poz_SM == 6)
                    {
//12.15 23.12.17 Зберегти
                         LCD_WriteString(lcd, 50 + 24*(poz_SM - 2),
115, "^^", &Font_12x20, col_b, col_b, LCD_SYMBOL_PRINT_FAST);
                         ScreenMode->Up(ScreenMode);
                         LCD_WriteString(lcd, 50 + 24*poz_SM, 115,
"^^", &Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);
                         return 0;
                    }
                    else //if(poz_SM < 6)
                    {
                         LCD_WriteString(lcd, 50 + 24*(poz_SM - 2),
115, "^^", &Font_12x20, col_b, col_b, LCD_SYMBOL_PRINT_FAST);
                         ScreenMode->Up(ScreenMode);
                         LCD_WriteString(lcd, 50 + 24*(poz_SM - 1),
115, "^^", &Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);
                         return 0;
                    }
               }
          }

          if (poz_SM == 9)              // якщо знаходимся на екрані
"Колірна схема"
          {
               fm->SetColor(fm, color_scheme[Screens[9]-
>Get_Poz(Screens[9])]);
               settingRAM->color_mode = Screens[9]-
>Get_Poz(Screens[9]);
               ScreenMode->Set_Poz(ScreenMode, 0);
               writeSPIFlash();// Запис на flash
               return 1;                                    // змінився
екран - потрібно надрукувати новий
          }

          if (poz_SM == 10)             // якщо знаходимся на екрані
"Демонстраційний режим"
          {
```

```
               if(Screens[10]->Get_Poz(Screens[10]) == 1)   //якщо
нажали "Демонстраційний режим"
               {
                    DEMO_ON();
               }
               else                                        //якщо нажали
"Робочий режим"
               {
                    DEMO_OFF();
               }
               ScreenMode->Set_Poz(ScreenMode, 0);
               writeSPIFlash();// Запис на flash
               return 1;                                   // змінився
екран - потрібно надрукувати новий
          }

          if (poz_SM == 11)   //якщо на екрані "Вибір режиму роботи
насоса сонячного колектора"
          {
               poz_S = Screens[11]->Get_Poz(Screens[11]);
               switch (poz_S)
               {
                    case 1:   // 1.     Вимкнено
                    case 2: // 2.  Примусова циркуляція при dT > 0°C
                    case 3: // 3.  Циркуляція при dT > 8°C
                         if(settingRAM->mode_flow != poz_S) // якщо
змінили режим насоса
                         {
                              settingRAM->mode_flow = poz_S; //
запам`ятовуємо новий режим
                              ScreenMode->Set_Poz(ScreenMode, 0);
     //то встановлюємо поточним екраном - інфо екран
                              writeSPIFlash();       // Збереження
на flash нових налаштувань
                              return 1;                // змінився
екран - потрібно надрукувати новий
                         }
                         break;
                    case 5:
                         printCor_dT();           // Друк екрану
"Налаштування dT колектора"
                         printCor_dT_Update();    // Оновлення
екрану "Налаштування dT колектора"
                         break;
               }
               return 0;
          }

          if (poz_SM == 12)             // якщо знаходимся на екрані
"Налаштування годинника"
          {
               settingRAM->calibrationRTC = Screens[12]-
>Get_Poz(Screens[12]);
```

```
                uint32_t var_calibr_RTC = abs(settingRAM-
>calibrationRTC);

                if(settingRAM->calibrationRTC == 0)
                {
                     if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_RESET, 0)
!= HAL_OK)
                     {
                          Error_Handler();
                     }
                }
                else if (settingRAM->calibrationRTC > 0)
                {
                     if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_SET,
var_calibr_RTC) != HAL_OK)
                     {
                          Error_Handler();
                     }
                }
                else
                {
                     if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_RESET,
var_calibr_RTC) != HAL_OK)
                     {
                          Error_Handler();
                     }
                }

                ScreenMode->Set_Poz(ScreenMode, 0);
                writeSPIFlash();// Запис на flash

                return 1;                                   // змінився
екран - потрібно надрукувати новий
           }

           if (poz_SM == 13)              // якщо знаходимся на екрані
"Коригування Т для запобіжного режиму"
           {
               settingRAM->T_max_valve = Screens[13]-
>Get_Poz(Screens[13]);   //запам`ятовуємо вибрану температуру
               ScreenMode->Set_Poz(ScreenMode, 0);
               writeSPIFlash();// Запис на flash
               return 1;
           }

           if (poz_SM == 14)              // якщо знаходимся на екрані
"Налаштування ТЕН"
           {
                poz_S = Screens[14]->Get_Poz(Screens[14]);
                switch (poz_S)
```

```
                {
                     case 1:   //HEATING_OFF  - ТЕН не вмикається
                     case 2: //HEATING_ON - ТЕН повинен вмикатись в
потрібні часові інтервали якщо темп.в баці (Твх) меньше заданої
                     case 3: // HEATING_ALL - ТЕН постійно підтримує
температуру
                          if(settingRAM->mode_heat != poz_S) // якщо
змінили режим ТЕН
                          {
                               settingRAM->mode_heat = poz_S; //
запам`ятовуємо новий режим
                               ScreenMode->Set_Poz(ScreenMode, 0);
     //встановлюємо поточним екраном - інфо екран
                               writeSPIFlash();       // Збереження
на flash нових налаштувань
                               return 1;                // змінився
екран - потрібно надрукувати новий
                          }
                          break;
                     case 5:
                          printCor_Time();    // Друк екрану
"Налаштування Time"
                          printCor_Time_Update();  // Оновлення
екрану "Налаштування Time"
                          break;
                }

                return 0;
           }

           if (poz_SM >= 16 && poz_SM <= 30)  // якщо знаходимся на
екрані "Налаштування TimePeriod"
           {
                //координати X, Y переміщення позицій символів "^^"
                uint16_t X[16] = {75, 160, 184, 232, 256, 75, 160,
184, 232, 256, 75, 160, 184, 232, 256, 130};
                uint16_t Y[16] = {85,  85,  85,  85,  85,115, 115,
115, 115, 115,145, 145, 145, 145, 145, 180};

                LCD_WriteString(lcd, X[poz_SM - 16], Y[poz_SM - 16],
"^^", &Font_12x20, col_b, col_b, LCD_SYMBOL_PRINT_FAST);
                ScreenMode->Up(ScreenMode);
                LCD_WriteString(lcd, X[poz_SM - 15], Y[poz_SM - 15],
"^^", &Font_12x20, col_f, col_b, LCD_SYMBOL_PRINT_FAST);

                //оновлюємо рамки таблиці
                LCD_DrawLineH(lcd, 0, 45, 319, fm->color_scheme-
>slider_color);
                LCD_DrawLineH(lcd, 0, 66, 319, fm->color_scheme-
>slider_color);
                LCD_DrawLineH(lcd, 0, 95, 319, fm->color_scheme-
>slider_color);
```

```
                LCD_DrawLineH(lcd, 0, 124, 319, fm->color_scheme-
>slider_color);
                LCD_DrawLineH(lcd, 0, 155, 319, fm->color_scheme-
>slider_color);
                LCD_DrawLineV(lcd, 140,45,110, fm->color_scheme-
>slider_color);
                LCD_DrawLineV(lcd, 215,45,110, fm->color_scheme-
>slider_color);

                return 0;
          }

          if(poz_SM == 31)
          {
                if(Screens[31]->Get_Poz(Screens[31]) == 1)   //якщо
нажали Так
                {    //   зберігаємо вибрані значення
                     settingRAM->Use_Period_1    = Screens[16]-
>Get_Poz(Screens[16]);
                     settingRAM->PeriodTime_1[0] = Screens[17]-
>Get_Poz(Screens[17]);
                     settingRAM->PeriodTime_1[1] = Screens[18]-
>Get_Poz(Screens[18]);
                     settingRAM->PeriodTime_1[2] = Screens[19]-
>Get_Poz(Screens[19]);
                     settingRAM->PeriodTime_1[3] = Screens[20]-
>Get_Poz(Screens[20]);

                     settingRAM->Use_Period_2    = Screens[21]-
>Get_Poz(Screens[21]);
                     settingRAM->PeriodTime_2[0] = Screens[22]-
>Get_Poz(Screens[22]);
                     settingRAM->PeriodTime_2[1] = Screens[23]-
>Get_Poz(Screens[23]);
                     settingRAM->PeriodTime_2[2] = Screens[24]-
>Get_Poz(Screens[24]);
                     settingRAM->PeriodTime_2[3] = Screens[25]-
>Get_Poz(Screens[25]);

                     settingRAM->Use_Period_3    = Screens[26]-
>Get_Poz(Screens[26]);
                     settingRAM->PeriodTime_3[0] = Screens[27]-
>Get_Poz(Screens[27]);
                     settingRAM->PeriodTime_3[1] = Screens[28]-
>Get_Poz(Screens[28]);
                     settingRAM->PeriodTime_3[2] = Screens[29]-
>Get_Poz(Screens[29]);
                     settingRAM->PeriodTime_3[3] = Screens[30]-
>Get_Poz(Screens[30]);

                     writeSPIFlash();
                }
```

```
                ScreenMode->Set_Poz(ScreenMode, 0);     //
повертаємось на інфо екран
                return 1;                               // змінився
екран - потрібно надрукувати новий
           }

           if (poz_SM == 32)             // якщо знаходимся на екрані
"Коригування dT колектора"
           {
               settingRAM->dT_min_pump = Screens[32]-
>Get_Poz(Screens[32]);   //запам`ятовуємо вибрану dT
               ScreenMode->Set_Poz(ScreenMode, 0);
               writeSPIFlash();// Запис на flash
               return 1;
           }

           if(poz_SM >= 33 && poz_SM <= 35)             // якщо
знаходимся на екрані "Робота насоса"
           {
                ScreenMode->Up(ScreenMode);
                print_Work_Pump_Update();
                return 0;
           }

           if(poz_SM == 36)
           {
                if(Screens[36]->Get_Poz(Screens[36]) == 1)   //якщо
нажали Зберегти
                {    //   зберігаємо вибрані значення
                     settingRAM->dT_OFF = Screens[33]-
>Get_Poz(Screens[33]);   // Гістерезис температури в градусах
                     settingRAM->MIN_ON_TIME = Screens[34]-
>Get_Poz(Screens[34]);   // мінімальна тривалість роботи насоса у
хвилинах
                     settingRAM->MIN_OFF_TIME = Screens[35]-
>Get_Poz(Screens[35]);   // мінімальна тривалість відпочинку насоса
у хвилинах

                     writeSPIFlash();
                }

                ScreenMode->Set_Poz(ScreenMode, 0);     //
повертаємось на інфо екран
                return 1;                               // змінився
екран - потрібно надрукувати новий
           }

           if(poz_SM == 37)//якщо знаходимся на екрані графіка
енергії
           {
                ScreenMode->Set_Poz(ScreenMode, 0);
                return 1;
           }
```

```
          if (poz_SM == 38)             // якщо знаходимся на екрані
"Вибір режиму роботи насоса басейна"
          {
               poz_S = Screens[38]->Get_Poz(Screens[38]);
               switch (poz_S)
               {
                    case 1:   //1.Спортивне плавання
                    case 2: //2.Відпочинок,купання дорослих
                    case 3: //3.Діти, сімейне купання
                    case 4: //4.Терапевтичні або спа-цілі
                    case 5: //5.Користувацький режим
                    case 6: //6.Насос вимкнено
                         if(settingRAM->mode_pump_pool != poz_S) //
якщо змінили режим
                         {
                              settingRAM->mode_pump_pool = poz_S; //
запам`ятовуємо новий режим

                              if(poz_S == 6) settingRAM->mode_heat =
HEATING_OFF;//якщо вимкнули насос басейна, то вимикаємо ТЕН

                              ScreenMode->Set_Poz(ScreenMode, 0);
     //встановлюємо поточним екраном - інфо екран
                              writeSPIFlash();       // Збереження
на flash нових налаштувань
                              return 1;                // змінився
екран - потрібно надрукувати новий
                         }
                         break;
                    case 8:
                         print_Modes_T_Pool();    // Друк екрану
"Налаштування температур режимів"
                         print_Modes_T_Pool_Update();  // Оновлення
екрану "Налаштування температур режимів"
                         break;
                    case 9:
                         printCor_dT_pool();           // Друк
екрану "Налаштування dT басейна"
                         printCor_dT_pool_Update();    // Оновлення
екрану "Налаштування dT басейна"
                         break;
               }

               return 0;
          }

          if(poz_SM >= 39 && poz_SM <= 48)             // якщо
знаходимся на екрані "Робота насоса"
          {
               ScreenMode->Up(ScreenMode);
               print_Modes_T_Pool_Update();
               return 0;
```

```
          }

          if(poz_SM == 49)
          {
               if(Screens[49]->Get_Poz(Screens[49]) == 1)   //якщо
нажали Зберегти
               {    //   зберігаємо вибрані значення
                    settingRAM->T_min_pool[0] = Screens[39]-
>Get_Poz(Screens[39]);
                    settingRAM->T_max_pool[0] = Screens[40]-
>Get_Poz(Screens[40]);
                    settingRAM->T_min_pool[1] = Screens[41]-
>Get_Poz(Screens[41]);
                    settingRAM->T_max_pool[1] = Screens[42]-
>Get_Poz(Screens[42]);
                    settingRAM->T_min_pool[2] = Screens[43]-
>Get_Poz(Screens[43]);
                    settingRAM->T_max_pool[2] = Screens[44]-
>Get_Poz(Screens[44]);
                    settingRAM->T_min_pool[3] = Screens[45]-
>Get_Poz(Screens[45]);
                    settingRAM->T_max_pool[3] = Screens[46]-
>Get_Poz(Screens[46]);
                    settingRAM->T_min_pool[4] = Screens[47]-
>Get_Poz(Screens[47]);
                    settingRAM->T_max_pool[4] = Screens[48]-
>Get_Poz(Screens[48]);
                    settingRAM->T_min_pool[5] = 19;
                    settingRAM->T_max_pool[5] = 20;
                    writeSPIFlash();
               }

               ScreenMode->Set_Poz(ScreenMode, 0);      //
повертаємось на інфо екран
               return 1;                                // змінився
екран - потрібно надрукувати новий
          }

          if (poz_SM == 50)              // якщо знаходимся на екрані
"Коригування dT басейна"
          {
              settingRAM->dT_min_pump_pool = Screens[50]-
>Get_Poz(Screens[50]);   //запам`ятовуємо вибрану dT
              ScreenMode->Set_Poz(ScreenMode, 0);
              writeSPIFlash();// Запис на flash
              return 1;
          }
     }
     return 0;
}

//функція відображає новий інфо екран
void Goto_New_Window(void)
```

```
{
     if(ScreenMode->Get_Poz(ScreenMode) ==   0)   //якщо на одному з
інфо екранів
     {
          LCD_Fill(lcd, fm->color_scheme->bg_color);
          switch (Screens[0]->Get_Poz(Screens[0]))     // в
залежності від номера екрана
          {
               case 0:   // інфо екран 0
                         printInfoStatic0(); // Друк статичної
картинки екрана 0
                         printInfoUpdate0(); // Друк значень на
екрані 0
                         break;
               case 1:   // інфо екран 1
                         printInfoStatic1(); // Друк статичної
картинки екрана 1
                         printInfoUpdate1(); // Друк значень на
екрані 1
                         break;
               case 2:   // інфо екран 2
                         printInfoStatic2();      // Друк статичної
картинки екрана 2
                         printInfoUpdate2(); // Друк значень на
екрані 2
                         break;
               case 3: // інфо екран 3
                         printChart3(); // Друк графіків
                         break;
               case 4: // інфо екран 4
                         printHistoryLog4(); // Друк журнала історії
                         break;
          }
     }
}

int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_DMA_Init();
  MX_RTC_Init();
  MX_SPI2_Init();
  MX_SPI1_Init();
  MX_TIM1_Init();
  MX_TIM3_Init();
  MX_TIM10_Init();
  MX_ADC1_Init();

  // Вимкнення переривання для EXTI0 (PA0) перед ініціалізацією:
  __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_0);
```

```
    HAL_NVIC_DisableIRQ(EXTI0_IRQn);

  // очищаємо ознаку, щоб перший раз таймер htim1 не запустився
відразу з запуском МК
    __HAL_TIM_CLEAR_FLAG(&htim1, TIM_SR_UIF);

chart_dry = CHART_DRY_New();       // Структура для графіків - інфо
екрани 2 та 3. Значення за останні 24 години
settingRAM = DataForSaveNew();

CoolData  = 0;
tick_SPI_Flash = 0;
FlashStorage_Init(&hspi1, MEM_CS_GPIO_Port, MEM_CS_Pin, chart_dry,
settingRAM);

  //nrf24
  csn_high();
  ce_high();
  HAL_Delay(5);
  ce_low();
  nrf24_init();
  nrf24_listen();
  nrf24_auto_ack_all(auto_ack);
  nrf24_en_ack_pld(disable);
  nrf24_en_dyn_ack(disable);
  nrf24_dpl(disable);
  nrf24_set_crc(en_crc, _1byte);
  nrf24_tx_pwr(n12dbm);
  nrf24_data_rate(_1mbps);
  nrf24_set_channel(40);
  nrf24_set_addr_width(5);
  nrf24_set_rx_dpl(0, disable);
  nrf24_pipe_pld_size(0, PLD_SIZE);
  nrf24_auto_retr_delay(4);
  nrf24_auto_retr_limit(10);
  nrf24_open_tx_pipe(tx_addr);
  nrf24_open_rx_pipe(0, tx_addr);
  ce_high();
  /* ---------------------------- Налаштування кнопок --------------
----------------- */
   KEYB_Add_Button(KEY_LEFT_GPIO_Port, KEY_LEFT_Pin, KEYB_LEFT,
KEYB_BUTTON_ACTIVE);
   KEYB_Add_Button(KEY_RIGHT_GPIO_Port, KEY_RIGHT_Pin, KEYB_RIGHT,
KEYB_BUTTON_ACTIVE);
   KEYB_Add_Button(KEY_UP_GPIO_Port, KEY_UP_Pin, KEYB_UP,
KEYB_BUTTON_ACTIVE);
   KEYB_Add_Button(KEY_DOWN_GPIO_Port, KEY_DOWN_Pin, KEYB_DOWN,
KEYB_BUTTON_ACTIVE);
   /* ---------------------------- Налаштування дисплея ------------
-------------------- */
   LCD_DMA_TypeDef dma_tx = { DMA1, LL_DMA_STREAM_4 };
   LCD_BackLight_data bkl_data = { TIM3, LL_TIM_CHANNEL_CH1, 0, 0,
50 };
```

```
   LCD_SPI_Connected_data spi_con = { SPI2, dma_tx,
                    LCD_RES_GPIO_Port, LCD_RES_Pin,
                        LCD_DC_GPIO_Port, LCD_DC_Pin,
                        LCD_CS_GPIO_Port, LCD_CS_Pin };

#ifndef LCD_DYNAMIC_MEM
   LCD_Handler lcd1;
#endif

   //для дисплея на контролері ili9341
   LCD = LCD_DisplayAdd( LCD,
 #ifndef  LCD_DYNAMIC_MEM
                                &lcd1,
 #endif
                                240, 320,
                                ILI9341_CONTROLLER_WIDTH,
ILI9341_CONTROLLER_HEIGHT,
                                PAGE_ORIENTATION_LANDSCAPE_MIRROR,
                                ILI9341_Init, ILI9341_SetWindow,
                                ILI9341_SleepIn, ILI9341_SleepOut,
                                &spi_con, LCD_DATA_16BIT_BUS,
bkl_data  );

     lcd = LCD; //вказівник на дисплей
     LCD_Init(lcd);
     EKRAN_ON();                          // увімкнення дисплея
     printStartInfo();// друк стартового напису

//   присвоюємо початкові значення структурам
     packet = PacketNew();
     ave_tIn = AVARAGE_New(3);
     ave_tOut = AVARAGE_New(3);
     ave_tPool = AVARAGE_New(3);
     ave_V = AVARAGE_New(3);

#if  !DEBUG
  HAL_IWDG_Refresh(&hiwdg);   // Скидання сторожового таймера
#endif
  readSPIFlash();   // читаємо файли з flash: settingRAM.dat -
структура з параметрами settingRAM, chart_dry.dat - структура з
даними графіків chart_dry

//застосовуються значення для калібрування годинника RTC
     uint32_t var_calibr_RTC = abs(settingRAM->calibrationRTC);
     if(settingRAM->calibrationRTC == 0)
     {
          if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_RESET, 0)
!= HAL_OK)
          {
               Error_Handler();
          }
     }
```

```
     else if (settingRAM->calibrationRTC > 0)
     {
          if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_SET,
var_calibr_RTC) != HAL_OK)
          {
               Error_Handler();
          }
     }
     else
     {
          if (HAL_RTCEx_SetSmoothCalib(&hrtc,
RTC_SMOOTHCALIB_PERIOD_32SEC, RTC_SMOOTHCALIB_PLUSPULSES_RESET,
var_calibr_RTC) != HAL_OK)
          {
               Error_Handler();
          }
     }

#if  !DEBUG
     HAL_IWDG_Refresh(&hiwdg);     // Скидання сторожового таймера
#endif
     fm = FileManagerNew();                         // Створення
обробника файлового менеджера

     fm->SetColor(fm, color_scheme[settingRAM->color_mode]);//
встановлюємо "Колірну схему" зчитану з flash

     if(settingRAM->demo_mode == 0)     //якщо demo-режим вимкнено
     {
          DEMO_OFF();          // то період виводу однієї точки
графіка = 12 хв. = 720 сек.
     }
     else //в режимі DEMO швидше: нова точка на гріфіку кожні 3 сек.
     {
          DEMO_ON();
     }

     get_time(1);   // отримання актуальних дати, часу і формування
масиву TimeBuf для виводу

#if  !DEBUG
     HAL_IWDG_Refresh(&hiwdg);     // Скидання сторожового таймера
перед читанням файлу
                          // щоб не було перевантаження МК у випадку
проблем з доступом до файлу
#endif

     TIM10->DIER |= TIM_DIER_UIE;  //дозволяємо переривання по
оновленню таймера TIM10
     TIM10->CR1  |= TIM_CR1_CEN;   //вмикаємо таймер TIM10 для
обробки натискання кнопок
```

```
     //запускаємо таймер htim1 в режимі переривання для зчитування
значень з датчика
     HAL_TIM_Base_Start_IT(&htim1);

     // Очистити можливі очікуючі переривання:
     __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_0);

     // Увімкнути переривання:
     HAL_NVIC_EnableIRQ(EXTI0_IRQn);

//   перевірка, чи працює NRF24
     uint8_t test_nrf24 = nrf24_is_connected(tx_addr, 5);
    if (test_nrf24 > 0)
     {
#if DEBUG
          printf("NRF24 ERROR!\r\n");
#endif
          LCD_WriteString(lcd, 10, 10, utf8_to_win1251("NRF24 не
знайдений або не відповідає!", OutputBuf), &Font_12x20, fm-
>color_scheme->text_color_dir, fm->color_scheme->bg_line2_color,
LCD_SYMBOL_PRINT_FAST);

          switch (test_nrf24)
          {
               case  1:
                    LCD_WriteString(lcd, 10, 30,
utf8_to_win1251("SPI не відповідає!", OutputBuf), &Font_12x20, fm-
>color_scheme->text_color_dir, fm->color_scheme->bg_line2_color,
LCD_SYMBOL_PRINT_FAST);
#if DEBUG
                    printf("SPI error!\r\n");
#endif
                    break;
               case  2:
                    LCD_WriteString(lcd, 10, 30,
utf8_to_win1251("TX_ADDR не збігається!", OutputBuf), &Font_12x20,
fm->color_scheme->text_color_dir, fm->color_scheme->bg_line2_color,
LCD_SYMBOL_PRINT_FAST);
#if DEBUG
                    printf("TX_ADDR mismatch!\r\n");
#endif
                    break;
               case  3:
                    LCD_WriteString(lcd, 10, 30,
utf8_to_win1251("RX0_ADDR не збігається!", OutputBuf), &Font_12x20,
fm->color_scheme->text_color_dir, fm->color_scheme->bg_line2_color,
LCD_SYMBOL_PRINT_FAST);
#if DEBUG
                    printf("RX0_ADDR mismatch!\r\n");
#endif
                    break;
          }
```

```
        while (1); // стоп
    } else {
#if DEBUG
        printf("NRF24 Ok!\r\n");
#endif
    }

     ErrorBuf[0] = 0;
     nrf24_listen();
     system_ready = 1;
     InsertIvent(8);

     while (1)
     {
          if(irq_RX == 1)
          {
               readNRF24();        //отримання параметрів з датчиків
          }

          if(irq_3sec == 1)
          {
               irq_3sec = 0;
               if(Start_Info_time > 0) printStartInfo(); // друк
стартового напису
               if(temp_ready == 1 ) ToDo_every_3sec(); // виконуємо,
якщо отримали хоч один раз дані
          }

          if (KEYB_kbhit() == 1)                         //якщо була
натиснута будь-яка кнопка
          {
               int8_t rslt;
               rslt = button_click_handler();               //то
запуск обробника натискання кнопок
               if(rslt == 1)                        //якщо є перехід
на новий екран,
               Goto_New_Window();            //то відображаємо його
          }

          if (FLAG_EKRAN_CHECK)                    //якщо дисплей
увімкнуто
          {
               if (packet_ready == 1)        //і якщо було оброблено
дані з датчиків
               {
                    packet_ready = 0;             //скидаємо ознаку
«дані оброблено»
                    if(ScreenMode->Get_Poz(ScreenMode) == 0)// якщо
знаходимся на якомусь інфо екрані
                    {
                         switch (Screens[0]->Get_Poz(Screens[0]))
                         { // оновлюємо зображення в залежності від
номера екрана
```

```
                                case 0: // якщо інформаційний екран 0
                                     printInfoUpdate0(); // Друк
значень на екрані 0
                                     break;
                                case 1: // якщо інформаційний екран 1
                                     printInfoUpdate1(); // Друк
значень на екрані 1
                                     break;
                                case 2: // якщо інформаційний екран 2
                                     printInfoUpdate2(); // Друк
значень на екрані 2
                                     break;
                                case 3: //якщо інформаційний екран 3
                                     updateChart3(); //оновлення
графіків
                                     break;
                                case 4: //якщо інформаційний екран 4
                                     UpdateHistoryLog4(); // Друк
журнала історії
                                     break;
                           }
                      }
                 }
            }
       }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

    if(system_ready == 0) return; // Ігнорувати події до завершення
ініціалізації

    if(GPIO_Pin == GPIO_PIN_0) // IRQ від nRF24
    {
        uint8_t status = nrf24_r_status();

        if (status & (1 << RX_DR)) {    // Дані отримано
               irq_RX = 1;
        } else if (status & (1 << TX_DS)) { // Дані передано успішно
               error_TX = 0;       //скидаємо кількість невдалих
передавань
        } else if (status & (1 << MAX_RT)) {    // Помилка:
досягнуто межі автоматичних повторів
            // підрахунок кількості помилок при передачі
               error_TX = (error_TX + 1) % 65535; //змінна error_TX
змінюється від 0 до 65534.
               nrf24_clear_max_rt(); // Очистити прапорець
             nrf24_flush_tx();     // Очистити TX FIFO
           //після помилки переходимо в пріоритений режим: отримання
             nrf24_listen();
        }
    }
}
```

ДОДАТОК В

Код програмного забезпечення

```
/* USER CODE BEGIN Header */
/**********************************************************
  * @file  : main.c
  * @brief : блок 2 - МК STM32F103 - версія від 06.12.2025
  **********************************************************/
/* Includes ------------------------------------------*/
#include "main.h"

#define DEBUG 1
#include <string.h>
#include "stdlib.h"
#include "NRF24.h"
#include "NRF24_reg_addresses.h"
#include <stdio.h>
#include "ssd1306.h"
#include <OneWire.h>

/* Private variables ---------------------------------------*/
I2C_HandleTypeDef hi2c2;
IWDG_HandleTypeDef hiwdg;
SPI_HandleTypeDef hspi1;
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart1;

//функція для відладки по SWO. Використовуємо: printf("t=%d",t);
int _write(int file, uint8_t *ptr, int len) {
    for (int DataIdx = 0; DataIdx < len; DataIdx++) {
        ITM_SendChar(*ptr++);
    }
    return len;
}

//   вибір типу реле:    =  0,  якщо  реле  комутується  при  подачі
низького сигналу
#define relay_high  1  //=  1,  якщо  реле  комутується  при  подачі
високого сигналу

//#define YF_S201 450.0f //  YF-S201  ~  450  імп/л  (залежить  від
калібрування!)
#define  YF_S201  90.0f//  щоб  імітувати  витрату  великого  насоса,
зменшуємо коєф. в 5 разів

#if relay_high//вибір типу реле
//   реле комутується при подачі високого сигналу
     #define RELAY_ON GPIO_PIN_SET
```

```
    #define RELAY_OFF GPIO_PIN_RESET
#else
//   реле комутується при подачі низького сигналу
    #define RELAY_ON GPIO_PIN_RESET
    #define RELAY_OFF GPIO_PIN_SET
#endif

uint8_t        Flags;    // байт ознак
               //    0    біт    -    насос    сонячного    колектора
увімкнено/вимкнено
               // 1 біт - ТЕН увімкнено/вимкнено
               // 2 біт - насос басейна увімкнено/вимкнено
               // 4 біт - демонстраційний режим увімкнено/вимкнено

#define PUMP_BIT         0   //   біт   увімкнення   насоса   сонячного
колектора в Flags
#define HEAT_BIT         1 // біт увімкнення ТЕН в Flags
#define PUMPPOOL_BIT     2 // біт увімкнення насоса басейна
#define VALVE_BIT        3 // біт відкриття клапана
#define DEMO_BIT         4 // біт увімкнення демонстраційного режиму

#define FLAG_PUMP_CHECK       ((Flags  &  (1u<<PUMP_BIT))  !=  0)     //
біт насоса перевірити на 1
#define      PUMP_APPLY_STATE()                                  do      {
HAL_GPIO_WritePin(PIN_PUMP_GPIO_Port,  PIN_PUMP_Pin,  (FLAG_PUMP_CHECK
? RELAY_ON : RELAY_OFF));  } while (0)//вмикаємо або вимикаємо насос
в залежності від біта
#define  PUMP_OFF()       do  {   HAL_GPIO_WritePin(PIN_PUMP_GPIO_Port,
PIN_PUMP_Pin,  RELAY_OFF);   } while  (0)//вимикаємо  насос  у  випадку
відмови датчика температури

#define FLAG_HEAT_CHECK       ((Flags  &  (1u<<HEAT_BIT))  !=  0)     //
біт ТЕН перевірити на 1
#define      HEAT_APPLY_STATE()                                  do      {
HAL_GPIO_WritePin(PIN_HEAT_GPIO_Port,  PIN_HEAT_Pin,  (FLAG_HEAT_CHECK
? RELAY_ON : RELAY_OFF));  } while (0)
#define  HEAT_OFF()       do  {   HAL_GPIO_WritePin(PIN_HEAT_GPIO_Port,
PIN_HEAT_Pin, RELAY_OFF);  } while (0)

#define FLAG_PUMPPOOL_CHECK   ((Flags   &   (1u<<PUMPPOOL_BIT))   !=   0)
// біт насоса басейна перевірити на 1
#define     PUMPPOOL_APPLY_STATE()                                  do     {
HAL_GPIO_WritePin(PUMP_POOL_GPIO_Port,                    PUMP_POOL_Pin,
(FLAG_PUMPPOOL_CHECK ? RELAY_ON : RELAY_OFF)); } while (0)
#define      PUMPPOOL_OFF()                                       do      {
HAL_GPIO_WritePin(PUMP_POOL_GPIO_Port,  PUMP_POOL_Pin,  RELAY_OFF);  }
while (0)

#define FLAG_VALVE_CHECK ((Flags  &  (1u<<VALVE_BIT))  !=  0)     // біт
клапана на 1
```

```
#define      VALVE_APPLY_STATE()                                    do      {
HAL_GPIO_WritePin(VALVE_GPIO_Port,   VALVE_Pin,   (FLAG_VALVE_CHECK   ?
RELAY_ON : RELAY_OFF)); } while (0)
#define  VALVE_OFF()         do  {   HAL_GPIO_WritePin(VALVE_GPIO_Port,
VALVE_Pin, RELAY_OFF); } while (0)

#define FLAG_DEMO_CHECK       ((Flags  &  (1u<<DEMO_BIT))  !=  0)      //
біт демо-режиму перевірити на 1

//для відображення на екрані статусу роботи пристроїв: on чи off
#define STATE_STR(b)       ((b) ? "on" : "off")
#define STR_PUMP_STATE     STATE_STR(FLAG_PUMP_CHECK)
#define STR_HEAT_STATE     STATE_STR(FLAG_HEAT_CHECK)
#define STR_PUMPPOOL_STATE   STATE_STR(FLAG_PUMPPOOL_CHECK)

extern   float   Temp[MAXDEVICES_ON_THE_BUS];//масив   для   збереження
температур датчиків
extern   uint8_t   sensorStatus[MAXDEVICES_ON_THE_BUS];   //масив   для
збереження стану датчиків
// 0 = OK
// 1 = not present
// 2 = ROM CRC error
// 3 = scratchpad CRC error
// 4 = invalid temperature (85 / -127)
// 5 = read error

#define ERROR_SENSOR  INT16_MAX    //    значення    для    передачі
непрацездатності датчика температури
//для   передачі   коду   помилки   передаєм   значення   INT16_MAX   -
sensorStatus[MAXDEVICES_ON_THE_BUS]

#define TIME_SCAN_SENSOR  3 // Період опитування датчиків, сек.
#define TIME_LOST_NRF  300     //час  (у  сек)  очікування  у  випадку
втати радіозвязку, після якого вимикаються пристрої

volatile int8_t irq_3sec =    0;             //   =1   -   ознака,   що
минуло 3 сек
volatile uint8_t irq_RX = 0; //ознака отримання даних
volatile  uint32_t  pulse_count  =  0;       //  Лічильник  імпульсів
витратоміра
volatile  float  flow_rate  =  0.0;  //  L/min  -  значення  витрати
теплоносія в сонячному колекторі
volatile uint16_t error_RX = 2;    //кількість    невдалих    отримань
даних радіомодулем. стартове значення =2 щоб змоделювати, що спочатку
звязку не було
volatile uint16_t error_TX = 2;    //кількість невдалих передач даних
радіомодулем
volatile uint16_t error_RX_miss = 2;    //кількість         пропущених
отримань даних радіомодулем

#define PLD_SIZE 8  //розмір пакета передавання/приймання в байтах
```

```
uint8_t  tx_addr[5]  =  {0x11,  0xF0,  0x34,  0x35,  0x54};//адреса
радіомодуля
uint8_t dataTX[PLD_SIZE];
uint8_t dataRX[PLD_SIZE];

//  Ознака завершення ініціалізації , якщо = 1
uint8_t system_ready = 0; //  = 0, якщо ще не завершилась початкова
ініціалізація

// структура Packet_TX для збереження поточних значень
typedef struct Packet_TX
{
     int16_t   t1; // поточна температура на вході в бак - в сотих
градуса
     int16_t   t2; // поточна температура на виході з бака - в сотих
градуса
     int16_t  t3; // поточна температура в басейні - в сотих градуса
     int16_t   V; // кількість імпульсів датчика витрати за останні
три секунди
}    Packet_TX;

Packet_TX packet_TX = { .t1 = 2500, .t2 = 2500,  .t3 = 2500, .V = 100
};
Packet_TX packet2_TX = { .t1 = 2500, .t2 = 2500, .t3 = 2500, .V = 100
};

typedef struct Packet_RX
{
    uint8_t flags;
    uint8_t payload[PLD_SIZE - 1];//для заповнення решти структури
} Packet_RX;

Packet_RX packet_RX = {
    .flags   = 0x00,
    .payload = {0}
};

//---------------------TX---------------------------
// Функція перетворення структури в масив байтів
void   packetToByteArray_TX(volatile   Packet_TX   *packet0,   uint8_t
*byteArray) {
  // Записуємо температуру (2 байти)
    byteArray[0] = (uint8_t)(packet0->t1 & 0xFF);         // молодший
байт
    byteArray[1] = (uint8_t)((packet0->t1 >> 8) & 0xFF); // старший
байт

    // Записуємо температуру (2 байти)
    byteArray[2] = (uint8_t)(packet0->t2 & 0xFF);         // молодший
байт
```

```
    byteArray[3] = (uint8_t)((packet0->t2 >> 8) & 0xFF); // старший
байт

    // Записуємо температуру (2 байти)
    byteArray[4] = (uint8_t)(packet0->t3 & 0xFF);         // молодший
байт
    byteArray[5] = (uint8_t)((packet0->t3 >> 8) & 0xFF); // старший
байт

    // Записуємо кількість імпульсів датчика витрати (2 байти)
    byteArray[6] = (uint8_t)(packet0->V & 0xFF);          // молодший
байт
    byteArray[7] = (uint8_t)((packet0->V >> 8) & 0xFF); // старший
байт
}

void BufferToPacket_TX(uint8_t* buffer, Packet_TX *packet0)
{
    if (packet0 == NULL) return;

    // Перетворення little-endian байтів у int16_t
    packet0->t1 = (int16_t)((buffer[1] << 8) | buffer[0]);
    packet0->t2 = (int16_t)((buffer[3] << 8) | buffer[2]);
    packet0->t3 = (int16_t)((buffer[5] << 8) | buffer[4]);
    packet0->V = (int16_t)((buffer[7] << 8) | buffer[6]);
}

//---------------------RX---------------------------
// Перетворення структури в масив байтів
static inline void packetToByteArray_RX(const Packet_RX *pkt, uint8_t
*byteArray) {
    // просто копіюємо всю пам’ять структури
    memcpy(byteArray, pkt, PLD_SIZE);
}

// Перетворення масиву байтів у вже існуючу структуру.
// out_pkt повинен бути валідним покажчиком на Packet_RX.
static inline void BufferToPacket_RX(const uint8_t *buffer, Packet_RX
*out_pkt) {
    memcpy(out_pkt, buffer, PLD_SIZE);
}

//обробник переривань таймера
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
     if(system_ready == 0) return; // Ігнорувати події до завершення
ініціалізації
     if(htim->Instance == TIM1) //перевіряємо чи переривання від 1-го
таймера TIM1
     {
         HAL_IWDG_Refresh(&hiwdg); // Скидання сторожового таймера
```

```
          irq_3sec = 1;   //встановлюємо ознаку «минуло 3 сек»

           error_RX_miss++;

          // YF-S201 ~ 450 імп/л (залежить від калібрування!)
//        flow_rate = (pulses * 60.0f) / 450.0f; // L/min    //якщо
переривання раз в секунду
//         так як у нас раз у три секунди то ділимо на 20, а не на 60
          flow_rate = (pulse_count * 20.0f) / YF_S201;
          pulse_count = 0;
      }
}

void readNRF24(void) //отримання даних з радіомодуля NRF24L01
{
    irq_RX = 0;
    if(nrf24_data_available())
    {
        uint8_t status = nrf24_r_status();
        if (status & (1 << RX_DR))  // Дані отримані успішно
        {
            nrf24_receive(dataRX, PLD_SIZE);
            BufferToPacket_RX(dataRX, &packet_RX);
            Flags = packet_RX.flags;
            error_RX = 0;          //скидаємо    кількість    невдалих
отримань

            if(sensorStatus[1]  ==  0  &&  sensorStatus[2]  ==  0  &&
sensorStatus[0] == 0)//якщо датчики працюють
            {
                //отримали нові дані, можливо потрібно змінити стан
пристроїв:
                PUMP_APPLY_STATE();
                HEAT_APPLY_STATE();
                PUMPPOOL_APPLY_STATE();
                VALVE_APPLY_STATE();
            }
            else//якщо є непрацездатність датчика
            {
                PUMP_OFF();
                HEAT_OFF();
                PUMPPOOL_OFF();
                VALVE_OFF();
            }
        }
        else
        { // Помилка при отриманні
#if DEBUG
            printf("ERROR RX!\n");
#endif
```

```
                // підрахунок кількості помилок при отриманні
                error_RX = (error_RX + 1) % 65535; //змінна    error_RX
змінюється від 0 до 65534.

            nrf24_clear_rx_dr(); // Очистити прапорець
            nrf24_flush_rx(); // Очистити буфер

            nrf24_stop_listen();
            HAL_Delay(300);
            nrf24_listen();
        }
          error_RX_miss     =   0;   //скидаємо   кількість   пропущених
отримань
    }
}

// структури та функції для режиму ДЕМО: генерація підвищення та
зниження температури
typedef struct
{
    int value;
    int direction;
    int min;
    int max;
    int step_min;
    int step_max;
} TriangleGenRand;

void TriangleGenRand_Init(TriangleGenRand *gen, int min, int max, int
step_min, int step_max)
{
    gen->min = min;
    gen->max = max;
    gen->value = min;
    gen->direction = 1;
    gen->step_min = step_min;
    gen->step_max = step_max;
}

int TriangleGenRand_Next(TriangleGenRand *gen)
{
    int range = gen->step_max - gen->step_min + 1;
    int step = (rand() % range) + gen->step_min;
    gen->value += gen->direction * step;
    if (gen->value >= gen->max)
    {
        gen->value = gen->max;
        gen->direction = -1;
    }
    if (gen->value <= gen->min)
    {
```

```
        gen->value = gen->min;
        gen->direction = 1;
    }
    return gen->value;
}

int TriangleGenRand_Next_Delta(TriangleGenRand *gen)
{
    int range = gen->step_max - gen->step_min + 1;
    int step = (rand() % range) + gen->step_min;
    gen->value += gen->direction * step;
//коли працює насос, то дельта зменшується і навпаки
     if(FLAG_PUMP_CHECK) gen->direction = -1;
     else gen->direction = 1;

     if (gen->value >= gen->max)
    {
        gen->value = gen->max;
        gen->direction = -1;
    }
    if (gen->value <= gen->min)
    {
        gen->value = gen->min;
        gen->direction = 1;
    }
    return gen->value;
}

int TriangleGenRand_Next_Delta2(TriangleGenRand *gen)
{
    int range = gen->step_max - gen->step_min + 1;
    int step = (rand() % range) + gen->step_min;
    gen->value += gen->direction * step;
//коли працює насос, то дельта зменшується і навпаки
     if(FLAG_PUMPPOOL_CHECK) gen->direction = -1;
     else gen->direction = 1;
     if (gen->value >= gen->max)
    {
        gen->value = gen->max;
        gen->direction = -1;
    }

    if (gen->value <= gen->min)
    {
        gen->value = gen->min;
        gen->direction = 1;
    }
    return gen->value;
}

int main(void)
```

```
{
     HAL_Init();
     SystemClock_Config();
  /* Initialize all configured peripherals */
     MX_GPIO_Init();
     MX_IWDG_Init();
     MX_SPI1_Init();
     MX_TIM1_Init();
     MX_I2C2_Init();
     MX_USART1_UART_Init();

  // очищаємо ознаку, щоб перший раз таймер htim1 не запустився
відразу з запуском МК
     __HAL_TIM_CLEAR_FLAG(&htim1, TIM_SR_UIF);
/* ----------- Ініціалізація дисплея ----------- */
     ssd1306_Init();
     ssd1306_FlipScreenVertically();
     ssd1306_Clear();
     ssd1306_SetColor(White);
/* ------ Налаштування радіопередавача NRF24 ----- */
     csn_high();
     ce_high();
     HAL_Delay(10);
     ce_low();
     nrf24_init();
     nrf24_listen();
     nrf24_auto_ack_all(auto_ack);
     nrf24_en_ack_pld(disable);
     nrf24_en_dyn_ack(disable);
     nrf24_dpl(disable);
     nrf24_set_crc(en_crc, _1byte);
     nrf24_tx_pwr(n12dbm);
     nrf24_data_rate(_1mbps);
     nrf24_set_channel(40);
     nrf24_set_addr_width(5);
     nrf24_set_rx_dpl(0, disable);
     nrf24_pipe_pld_size(0, PLD_SIZE);
     nrf24_auto_retr_delay(4);
     nrf24_auto_retr_limit(10);
     nrf24_open_tx_pipe(tx_addr);
     nrf24_open_rx_pipe(0, tx_addr);
     ce_high();

     HAL_IWDG_Refresh(&hiwdg);     // Скидання сторожового таймера
//   перевірка, чи працює NRF24
     uint8_t test_nrf24 = nrf24_is_connected(tx_addr, 5);
    if (test_nrf24 > 0)
     {
#if DEBUG
          printf("NRF24 ERROR!\r\n");
#endif
```

```
          ssd1306_Clear();
          ssd1306_SetCursor(0, 0);
          ssd1306_WriteString("NRF24 ERROR!", Font_11x18);
          ssd1306_SetCursor(0, 17);

          switch (test_nrf24)
          {
               case  1:
                    ssd1306_WriteString("SPI error!", Font_11x18);
#if DEBUG
                    printf("SPI error!\r\n");
#endif
                    break;
               case  2:
                    ssd1306_WriteString("TX_ADDR          mismatch!",
Font_7x10);
#if DEBUG
                    printf("TX_ADDR mismatch!\r\n");
#endif
                    break;
               case  3:
                    ssd1306_WriteString("RX0_ADDR          mismatch!",
Font_7x10);
#if DEBUG
                    printf("RX0_ADDR mismatch!\r\n");
#endif
                    break;
          }
          ssd1306_UpdateScreen();
          HAL_IWDG_Refresh(&hiwdg);     //    Скидання    сторожового
таймера

        while (1); // стоп
    } else {
#if DEBUG
        printf("NRF24 Ok!\r\n");
#endif
    }

     nrf24_listen();
     get_ROMid();   //отримуємо адреси датчиків температур
     Get_Temperatures();//отримуємо температури з датчиків

// для режиму ДЕМО: генерація підвищення та зниження температури
     srand(HAL_GetTick());  // ініціалізація rand()
    TriangleGenRand gen_Tin;
     //генерація  Tin  в  діапазоні  від  15  до  72  градусів.  Крок
зростання/зменшення випадковий: від 0.1 до 1.5
    TriangleGenRand_Init(&gen_Tin, 1500, 7200, 10, 150);
```

```
    //генерація різниці температур dT в діапазоні від 0 до 15 градусів. Крок зростання/зменшення випадковий: від 0 до 1
    TriangleGenRand gen_dT;
   TriangleGenRand_Init(&gen_dT, 0, 1500, 0, 100);

    TriangleGenRand gen_dT_pool;
   TriangleGenRand_Init(&gen_dT_pool, 0, 1000, 0, 50);

    Flags =  0x00;
    //запускаємо таймер htim1 в режимі переривання для зчитування значень з датчика
    HAL_TIM_Base_Start_IT(&htim1);
    system_ready = 1;

    while (1)
    {
        if(irq_RX == 1)          //якщо є переривання від NRF24L0
        {
            readNRF24();   //отримуємо дані з радіомодуля NRF24L01
        }

        if (irq_3sec == 1)  //якщо минуло 3 сек
        {
            irq_3sec = 0;  //скидаємо ознаку «минуло 3 сек»
            Get_Temperatures();//отримуємо температури з датчиків

            if(!FLAG_DEMO_CHECK)     //якщо немає ражиму Демо
            {
                // Читаємо температуру
                if(sensorStatus[1] == 0) packet_TX.t1 = (int16_t) (Temp[1] * 100);//якщо датчик T1 працює
                else packet_TX.t1      =      ERROR_SENSOR      - sensorStatus[1];    //якщо є непрацездатність датчика T1

                if(sensorStatus[2] == 0) packet_TX.t2 = (int16_t) (Temp[2] * 100);
                else packet_TX.t2      =      ERROR_SENSOR      - sensorStatus[2];

                if(sensorStatus[0] == 0) packet_TX.t3 = (int16_t) (Temp[0] * 100);
                else packet_TX.t3      =      ERROR_SENSOR      - sensorStatus[3];


              if(sensorStatus[1] == 0 && sensorStatus[2] == 0 && sensorStatus[0] == 0)//якщо всі датчики працюють
              {  //стан пристроїв приводиться у відповідність до Flags
                PUMP_APPLY_STATE();
                HEAT_APPLY_STATE();
```

```
                    PUMPPOOL_APPLY_STATE();
                    VALVE_APPLY_STATE();
                }
                else//якщо є непрацездатність хотя б одного датчика
                {//то вимикаємо пристрої
                    PUMP_OFF();
                    HEAT_OFF();
                    PUMPPOOL_OFF();
                    VALVE_OFF();
                }

                //якщо втрата радіозвязку більше ніж TIME_LOST_NRF
                if  ((int16_t)(error_RX_miss  *  TIME_SCAN_SENSOR)  >=
(int16_t)TIME_LOST_NRF)
                {//то вимикаємо пристрої
                    PUMP_OFF();
                    HEAT_OFF();
                    PUMPPOOL_OFF();
                    VALVE_OFF();
                }
            }
               else//якщо режим Демо
               {
                    packet_TX.t1 = TriangleGenRand_Next(&gen_Tin);
                    packet_TX.t2        =        packet_TX.t1        -
TriangleGenRand_Next_Delta(&gen_dT);
                    packet_TX.t3        =        packet_TX.t2        -
TriangleGenRand_Next_Delta2(&gen_dT_pool);// T в басейні
                    if(packet_TX.t3 > 3600) //якщо T в басейні вище
36 градусів
                    { packet_TX.t3 = 3600; } //то  значення  обмежуємо
36 градусами
               }

               // витрата у сотих л/хв
               packet_TX.V = (int16_t) (flow_rate * 100);

#if DEBUG
               printf("T1  =  %.2f  C\r\n",  Temp[1]);  //  Виводимо
температуру Твх
               printf("T2  =  %.2f  C\r\n",  Temp[2]);  //  Виводимо
температуру Твих
               printf("T3  =  %.2f  C\r\n",  Temp[0]);  //  Виводимо
температуру Тбас
               printf("V  =  %.2f  L/min\r\n",  flow_rate);  //  Виводимо
витрату у л/хв
#endif

               // Оновлення даних для передачі
               packetToByteArray_TX(&packet_TX, dataTX);
```

```
                nrf24_stop_listen();
                HAL_Delay(300);      //затримка щоб радіомодуль точно
змінив режим
                nrf24_transmit(dataTX, PLD_SIZE);  //       Передавання
даних

                // Обробка результату передачі
                if (error_TX > 1 || error_RX > 1 || error_RX_miss  >
1)
                {
                    //якщо є помилки при радіообміні то запалюємо
світлодіод на платі
                     HAL_GPIO_WritePin(LED_GPIO_Port,          LED_Pin,
GPIO_PIN_SET);

#if DEBUG
                    printf("Transmission failed: %d\n", error_TX);
                    printf("Receive failed: %d\n", error_RX);
#endif
                }
                else
                {
                     //якщо помилки відсутні радіообміні то тушимо
світлодіод на платі
                     HAL_GPIO_WritePin(LED_GPIO_Port,          LED_Pin,
GPIO_PIN_RESET);
                }

                //конвертуємо передані дані для виводу на екран
                BufferToPacket_TX(dataTX, &packet2_TX);

                //після передавання переходимо в режим очікування
отримання даних
                nrf24_clear_tx_ds(); // Очистити прапорець
                nrf24_listen();//радіомодуль   переходить   в   режим
приймання

        // Оновлення дисплея
                char str[27];
                int8_t vT1, vT2, vT3, fT1, fT2, fT3;

          // визначаємо цілу частину значень температур
          //якщо є проблеми з датчиком (sensorStatus не 0) то
температура буде показувати 0.0
                if(sensorStatus[1] == 0) vT1  =  (int)(packet2_TX.t1  /
100.0);
                else vT1 = 0;

                if(sensorStatus[2] == 0) vT2  =  (int)(packet2_TX.t2  /
100.0);
                else vT2 = 0;
```

```
                if(sensorStatus[0] == 0) vT3  =  (int)(packet2_TX.t3  /
100.0);
                else vT3 = 0;

           // визначаємо десяту частину значень температур
                if(sensorStatus[1] == 0) fT1                            =
(int)(10.0*((packet2_TX.t1 / 100.0) - vT1));
                else fT1 = sensorStatus[1];

                if(sensorStatus[2] == 0) fT2                            =
(int)(10.0*((packet2_TX.t2 / 100.0) - vT2));
                else fT2 = sensorStatus[2];

                if(sensorStatus[0] == 0) fT3                            =
(int)(10.0*((packet2_TX.t3 / 100.0) - vT3));
                else fT3 = sensorStatus[0];

                ssd1306_Clear();
                ssd1306_SetCursor(0, 0);
                sprintf(str, "Tin   Tout  Tpool");
                ssd1306_WriteString(str, Font_7x10);

                ssd1306_SetCursor(0, 10);
                sprintf(str, "%2d  %2d  %2d", vT1, vT2, vT3);
                ssd1306_WriteString(str, Font_11x18);

                ssd1306_SetCursor(22, 17);
                sprintf(str, "%d", fT1);
                ssd1306_WriteString(str, Font_7x10);

                ssd1306_SetCursor(66, 17);
                sprintf(str, "%d", fT2);
                ssd1306_WriteString(str, Font_7x10);

                ssd1306_SetCursor(110, 17);
                sprintf(str, "%d", fT3);
                ssd1306_WriteString(str, Font_7x10);

                ssd1306_SetCursor(0, 27);
                if(!FLAG_DEMO_CHECK)    sprintf(str,    "V:    %.2f",
packet2_TX.V / 100.0);
                else  sprintf(str,  "V:  %.1f  DEMO",  packet2_TX.V  /
100.0);
                ssd1306_WriteString(str, Font_11x18);

                ssd1306_SetCursor(0, 44);
                sprintf(str,   "pump-%s   heat-%s",   STR_PUMP_STATE,
STR_HEAT_STATE);
                ssd1306_WriteString(str, Font_7x10);
```

```
                ssd1306_SetCursor(0, 53);
                sprintf(str, "pump pool-%s", STR_PUMPPOOL_STATE);
                ssd1306_WriteString(str, Font_7x10);

                ssd1306_UpdateScreen();
            }
        }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(system_ready == 0) return; // Ігнорувати події до завершення
ініціалізації
    if(GPIO_Pin == GPIO_PIN_0) // IRQ від nRF24
    {
        uint8_t status = nrf24_r_status();

        if (status & (1 << RX_DR)) // Дані отримано
        {
            irq_RX = 1;
        } else if (status & (1 << TX_DS))// Дані передано успішно
          {
                error_TX = 0;       //скидаємо    кількість    невдалих
передавань
        } else if (status & (1 << MAX_RT)) { //   Помилка:   досягнуто
межі автоматичних повторів
                // підрахунок кількості помилок при передачі
                error_TX = (error_TX + 1) % 65535; //змінна    error_TX
змінюється від 0 до 65534.
                nrf24_clear_max_rt(); // Очистити прапорець
            nrf24_flush_tx();     // Очистити TX FIFO
            //після помилки переходимо в режим отримання
            nrf24_listen();
        }
    }
    if (GPIO_Pin == GPIO_PIN_1) // IRQ від витратоміра (PA1 -> EXTI1)
    { pulse_count++; }
}
```