Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп'ютерних наук

Кваліфікаційна робота магістра з
Теми: «Методи і засоби програмної підтримки вбудованої комп’ютеризованої системи для універсального зарядного пристрою»

Виконав:
здобувач вищої освіти 2 курсу KN1-M23 групи
спеціальності 122 Комп'ютерні науки
Кравчук В.О.
Керівник: Слободянюк О.В., к.т.н., доцент кафедри комп’ютерних наук,
Рецензент: Поведа Р.А.
к.ф-м.н., доцент кафедри фізики

Кам’янець-Подільський – 2024

ЗМІСТ

ВСТУП

Радіоелектроніка охоплює як теоретичні основи, так і практичні аспекти застосування електронних, іонних і напівпровідникових приладів у різних системах, пристроях та установках, що використовуються в багатьох галузях сучасної економіки. Завдяки високій гнучкості електронних пристроїв, їхній швидкодії, точності вимірювань і чутливості, відкриваються нові можливості для вдосконалення різних науково-технічних напрямків, зокрема в автоматизації процесів, телекомунікаціях, енергетиці та медицині.

Одним з основних елементів таких систем є зарядний пристрій, призначений для зарядки акумуляторних та конденсаторних батарей. Цей пристрій складається з джерела живлення (генератора або трансформатора) з випрямлячем струму, а також системи розподілу, яка включає регулятори напруги та автоматичні вимикачі. Потужність зарядного пристрою визначається параметрами акумуляторних батарей, такими як ємність, внутрішній опір та встановлена тривалість зарядки. Застосування зарядних пристроїв охоплює різні режими зарядки: періодичну, безперервну та переривчасту, що дозволяє оптимізувати процес зарядки в залежності від конкретних потреб системи.

Актуальність даної роботи полягає в розробці універсального зарядного пристрою на основі мікроконтролера Arduino та набору додаткових датчиків та електронних схемотехнічних компонентів. Використання Arduino у цій розробці також дає можливість провести попереднє моделювати та тестування процесу заряджання в реальному часі, що забезпечує високий рівень контролю та точності. Це особливо важливо в умовах швидкого розвитку відновлюваних джерел енергії та потреби в ефективних системах зберігання енергії, зокрема в акумуляторних технологіях.

Метою роботи є створення інтелектуального зарядного пристрою для різних типів акумуляторних батарей (АКБ) із застосуванням мікроконтролера. Пристрій призначений для заряджання свинцево-кислотних, AGM, GEL, літій-іон-

них, літій-титанатних і нікель-кадмієвих акумуляторів з напругою 3-24 В і ємністю 1-255 А·год, які використовуються в автомобілях, мотоциклах, катерах, скутерах, джерелах безперебійного живлення (ДБЖ) тощо.

Відповідно до мети було сформульовано такі задачі:

1. Розробити зарядний пристрій з мікроконтролером, який буде здатен підтримувати різні типи акумуляторів та оптимізувати процес заряду.

2. Реалізувати декілька алгоритмів заряду, включаючи попередній заряд для сильно розряджених акумуляторів, основний заряд за допомогою стабілізації струму та напруги, а також завершальні етапи заряду.

3. Реалізувати функції для контролю та тренування акумуляторів, таких як розряд, вимірювання внутрішнього опору та проведення контрольно-тренувальних циклів (КТЦ).

Забезпечення зручного інтерфейсу для відображення параметрів заряду та керування пристроєм за допомогою рідкокристалічного дисплея та енкодера з кнопками.

Предметом дослідження є засоби та технології розробки автоматизованих систем заряджання акумуляторних батарей.

Об’єктом дослідження є розробка архітектури та програмно апаратних комплексів заряджання акумуляторних батарей.

Практичне значення одержаних результатів: результатом роботи створення є повноцінної зарядної комп’ютеризованої системи з засобами її програмної підтримки.

Структура роботи. Дипломна робота магістра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ АРХІТЕКТУР СУЧАСНИХ ПРИСТРОЇВ ЗАРЯДЖАННЯ АКУМУЛЯТОРНИХ БАТАРЕЙ

1.1. Загальні відомості про зарядні пристрої

Ємність та тривалість роботи акумуляторних батарей значною мірою залежать від типу та якості зарядних пристроїв, які використовуються для їх зарядки. Зарядні пристрої впливають на вибір методу зарядки та режиму розрядки, що відіграє важливу роль у підтриманні ефективної роботи батарей. Вибір оптимального зарядного пристрою часто недооцінюється користувачами побутової техніки, проте це питання є критично важливим. Неправильний підхід до вибору зарядного пристрою може призвести до зниження ємності батарей, їх швидкого зносу або втрати здатності утримувати заряд. Інвестиції в якісний зарядний пристрій виправдовуються з точки зору подовження терміну служби акумуляторів та забезпечення їх ефективної роботи.

Зарядні пристрої розробляються на основі двох основних принципів зарядки: обмеження струму та обмеження напруги. Метод обмеження струму зазвичай застосовується для нікель-кадмієвих (NiCd) та нікель-металгідридних (NiMH) акумуляторів, тоді як для свинцево-кислотних (SLA), літій-іонних (Li-ion) та літій-полімерних (LiPo) батарей використовується метод обмеження напруги. Важливо враховувати ці характеристики, оскільки неправильний вибір методу зарядки може призвести до перегріву, втрати ємності або пошкодження батарей.

Зарядні пристрої, що працюють в режимі постійного струму (гальваностатичний режим), часто використовуються для зарядки лужних герметичних акумуляторів, таких як NiMH і NiCd. Ці пристрої забезпечують постійний струм протягом всього процесу зарядки, однак мають обмежене застосування, оскільки підходять лише для певних типів батарей. Найпростіші побутові зарядні пристрої, які використовуються для зарядки від 1 до 4 акумуляторів, переважно мають незмінний струм, і користувач самостійно визначає час зарядки, що може призвести до недостатнього або надмірного заряду батарей[24].

У більшості випадків ці пристрої живляться через трансформатор від стандартної мережі з напругою 220В і забезпечують випрямлений струм з низьким рівнем стабілізації. Однак, відсутність регулювання струму та залежність часу зарядки від користувача може негативно впливати на довговічність і продуктивність акумуляторних батарей. Тому важливо звертати увагу на технічні характеристики зарядних пристроїв і вибирати ті, що відповідають вимогам конкретного типу батарей, щоб забезпечити їх стабільну і тривалу роботу

Універсальність побутових зарядних пристроїв часто визначається їхньою здатністю приймати акумулятори різних типорозмірів і забезпечувати постійний зарядний струм, зазвичай на рівні 0,1 С (де С — це ємність акумулятора, виражена в ампер-годинах). Це значення струму розраховане для типових акумуляторів відповідного розміру, але при цьому важливо ретельно підбирати акумулятори для таких пристроїв і правильно визначати час їх зарядки, щоб уникнути перезарядки або недостатнього заряду.

За останні п'ять-сім років промисловість акумуляторів зазнала значного прогресу, що призвело до виробництва лужних акумуляторів однакових габаритів, але з ємністю, яка може відрізнятися в три і більше разів. У той же час, використання універсальних зарядних пристроїв для зарядки таких акумуляторів може призвести до тривалого процесу зарядки, який часто є неефективним через зарядний струм, який значно менший за рекомендований стандарт. Це пов'язано з тим, що зарядні пристрої з фіксованим струмом не враховують збільшену ємність сучасних акумуляторів, що може негативно вплинути на якість зарядки. Основною перевагою таких зарядних пристроїв є їх низька ціна, але їх використання для акумуляторів з великою ємністю може бути економічно неефективним через тривалий час зарядки та можливе погіршення характеристик акумулятора.

Зарядні пристрої для свинцево-кислотних (SLA), літій-іонних (Li-ion) та літій-полімерних (LiPo) акумуляторів потребують двоетапного процесу зарядки: на першій стадії забезпечується стабілізація струму (гальваностатичний режим), а на другій — стабілізація напруги (потенціостатичний режим). Важливим є та-

кож контроль завершення процесу зарядки, який може здійснюватися двома способами: або за фіксованим часом, або шляхом зниження струму до визначеного мінімального рівня. Належний контроль заряду є критично важливим, оскільки акумулятори чутливі до умов перезаряду, що може призвести до зниження їхньої ємності, теплового перегріву або навіть пошкодження.

Для нікель-кадмієвих (NiCd) та нікель-металгідридних (NiMH) акумуляторів існують три основні типи зарядних пристроїв:

1. Пристрої нормального (повільного) заряду: вони забезпечують невеликий зарядний струм і призначені для тривалого процесу зарядки. Такий метод мінімізує ризик перегріву та перевантаження акумуляторів, але вимагає значного часу для завершення циклу зарядки.

2. Пристрої швидкого заряду: забезпечують зарядку акумуляторів за допомогою вищого струму, що дозволяє скоротити час зарядки до кількох годин. Ці пристрої часто оснащені системами захисту від перегріву та автоматичними вимикачами для захисту акумуляторів.

3. Пристрої швидкісного заряду: такі пристрої забезпечують ще швидший процес зарядки за рахунок застосування високих струмів. Хоча час зарядки значно скорочується, вони потребують особливого контролю за температурою та станом акумуляторів, оскільки ризик їх пошкодження зростає.

Таким чином, вибір зарядного пристрою залежить не лише від його вартості, але й від типу акумулятора, що заряджається, його ємності та вимог до безпеки зарядки. Точний підбір відповідного зарядного пристрою дозволяє збільшити термін служби акумуляторів та забезпечити їх ефективну роботу без втрати ємності та ризику пошкодження.

Зарядні пристрої нормального (повільного) заряду зазвичай використовуються для зарядки нікель-кадмієвих (NiCd) акумуляторів. Ці пристрої, відомі також як нічні зарядні пристрої, працюють при зарядному струмі на рівні 0,1 C (де C — це номінальна ємність акумулятора, виражена в ампер-годинах). Тривалість заряду становить від 14 до 16 годин, що є типовим для низького струму зарядки. Основною

характеристикою таких пристроїв є відсутність індикатора завершення заряду, оскільки при малих струмах важко точно визначити момент закінчення зарядки. Вони є одними з найдоступніших за ціною, але застосовуються тільки для зарядки NiCd акумуляторів, оскільки їх функціональність є обмеженою.

Зарядні пристрої для більш сучасних нікель-металгідридних (NiMH) акумуляторів потребують більш точної стабілізації струму. При використанні таких зарядних пристроїв важливо правильно налаштувати зарядний струм, щоб уникнути перегріву акумулятора. Якщо зарядний струм встановлено коректно, акумулятор може залишатися трохи теплим на дотик після повної зарядки, і немає необхідності негайно відключати його від зарядного пристрою. Проте, рекомендується від'єднати батарею після завершення заряду для подовження терміну її служби. Якщо зарядний пристрій розрахований на зарядку більш потужних акумуляторів, а використовується для менших, це може призвести до перегріву акумулятора, що є сигналом про початок процесу теплового руйнування.

При відсутності можливості знизити струм або автоматично зупинити зарядку, в другій половині циклу можуть виникнути серйозні проблеми через надмірне нагрівання. Єдиний спосіб уникнути пошкодження акумулятора — це негайно відключити його, коли він стає занадто гарячим. Якщо ж використовується недостатньо потужний зарядний пристрій для зарядки великого акумулятора, він не досягне повної зарядки, що призведе до зниження ємності батареї.

Зарядні пристрої швидкого заряду зазвичай класифікуються як пристрої середнього класу, як за швидкістю заряду, так і за вартістю. Вони дозволяють заряджати акумулятори протягом 3-6 годин при зарядному струмі, який становить близько 0,3 С. У цих пристроях обов'язковим елементом є схема контролю напруги акумулятора, яка автоматично припиняє зарядку після досягнення певного рівня напруги. Це дозволяє уникнути ризику перезаряду і забезпечити більш точний процес зарядки. Завдяки такому підходу, зарядні пристрої швидкого заряду забезпечують кращий захист і обслуговування акумуляторів порівняно з повільними зарядними пристроями.

З розвитком технологій, пристрої швидкого заряду поступово поступилися місцем пристроям швидкісного заряду, які забезпечують ще більш швидкий і точний процес зарядки, що робить їх більш зручними для користувачів сучасних акумуляторних технологій. Швидкісні зарядні пристрої зазвичай оснащені додатковими механізмами захисту, що дозволяє збільшити ефективність зарядки без ризику пошкодження акумуляторів.

Зарядні пристрої швидкісного заряду відзначаються значними перевагами у порівнянні з іншими типами зарядних пристроїв. Основною перевагою є значно менший час заряду, що досягається завдяки використанню потужних джерел напруги та спеціальних систем контролю і управління процесом заряджання. Однак, ці пристрої є дорожчими через складність конструкції та необхідність забезпечення надійного контролю над процесом заряду.

Тривалість заряду у таких пристроях залежить від кількох факторів: сили зарядного струму, ступеня розряду акумуляторної батареї, її ємності та типу. Наприклад, при струмі заряду 1C (де 1C — це струм, який дорівнює номінальній ємності акумулятора, вираженій в ампер-годинах), розряджена нікель-кадмієва (NiCd) батарея може бути заряджена за менше ніж одну годину[7].

1.2 Основні методи зарядки акумулятора

Ємність акумулятора – це кількість заряду, яку може зберігати акумулятор, вимірюється в ампер-годинах (А·год) або міліампер-годинах (мА·год). Високоефективні зарядні пристрої також оснащені функцією автоматичного перемикання в режим підтримки заряду при досягненні повної зарядки. У цьому режимі струм заряду знижується до безпечного рівня, а таймер вимикає пристрій після певного часу для запобігання перегріву та пошкодження акумулятора[2].

Сучасні пристрої швидкісного заряду широко використовуються для зарядки нікель-металгідридних (NiMH) та нікель-кадмієвих (NiCd) акумуляторів. Однак важливо, щоб ці пристрої використовувалися тільки для акумуляторів, рекомендованих для швидкісного заряду, оскільки підвищений зарядний струм вимагає ретельного контролю. Порушення алгоритму заряду може призвести до перегріву та пошкодження акумуляторів.

Деякі сучасні батареї мають спеціальне електричне маркування, яке наноситься на заводах-виробниках. Це маркування дозволяє зарядному пристрою автоматично розпізнавати тип акумулятора та його основні електричні характеристики. Після цього зарядний пристрій встановлює оптимальний режим заряду, який враховує максимальну силу струму та алгоритм процесу, що є найбільш ефективним для конкретного типу батареї.

Варто зазначити, що алгоритми заряду свинцево-кислотних і літій-іонних (Li-ion) акумуляторів суттєво відрізняються від алгоритмів заряду NiCd і NiMH батарей. Неправильний вибір зарядного пристрою або алгоритму може призвести до значних втрат ємності акумулятора або навіть до його виходу з ладу.

Основні методи зарядки акумуляторів включають:

1. *Зарядка при постійному струмі (гальваностатичний режим):* Цей метод полягає в підтримці постійного значення струму протягом усього процесу зарядки. Найчастіше використовується для нікель-кадмієвих та нікель-металгідридних акумуляторів.

2. *Контрольно-тренувальний цикл***:** Це поєднання циклів розряду і заряду, яке дозволяє уникнути ефекту пам’яті в акумуляторах і продовжити їх термін служби.

3. *Зарядка імпульсним струмом***:** Процес зарядки проходить у вигляді коротких імпульсів струму, що дозволяє зменшити теплові втрати і покращити ефективність заряджання.

4. *Зарядка пульсуючим струмом***:** Використовується для швидкісної зарядки, де струм періодично змінюється між високими та низькими значеннями для покращення проникнення заряду всередину електродів.

5. *Зарядка асиметричним струмом***:** Цей метод зарядки використовує різні фази струму на етапі зарядки і розряду, що допомагає уникнути перезарядки та покращує стабільність акумулятора.

6. *Зарядка за правилом ампер-годин***:** Під час зарядки контролюється загальна кількість ампер-годин, що проходять через акумулятор, для точного визначення моменту завершення процесу.

Кожен із цих методів має свої переваги та недоліки і застосовується залежно від типу акумуляторів та специфічних умов експлуатації[17].

РОЗДІЛ 2. АНАЛІЗ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Порівняльний аналіз існуючих платформ та засобів

Сучасний ринок пропонує широкий вибір зарядних пристроїв, що відповідають різноманітним потребам користувачів. Вони розрізняються за функціональністю, ціною, підтримкою типів акумуляторів та можливістю налаштування. Існуючі рішення можна умовно поділити на універсальні зарядні пристрої, спеціалізовані системи та пристрої з інтегрованими контролерами.

Спеціалізовані зарядні пристрої

Серед популярних рішень виділяються пристрої, побудовані на інтегрованих контролерах, які забезпечують базові алгоритми заряджання:

Контролер BQ24133 (Texas Instruments): Цей інтегральний зарядний контролер призначений для роботи з Li-ion акумуляторами та підтримує кілька етапів заряджання:

- Передзарядка: низький струм заряджання для відновлення сильно розрядженого акумулятора.
- Швидке заряджання: стабільний струм до досягнення певного рівня напруги.
- Стабілізація: заряджання при постійній напрузі до повного заряджання.

  BQ24133 є ефективним рішенням для портативних пристроїв, таких як ноутбуки та мобільні телефони. Проте його недолік полягає у відсутності можливості адаптації алгоритму до специфічних потреб користувача.

2. Контролер TP4056:

   Простий у використанні контролер для заряджання Li-ion акумуляторів. Він забезпечує стабільний струм і напругу, має вбудовані функції захисту від перевантаження та перегріву. Основними перевагами TP4056 є низька

вартість і мінімум зовнішніх компонентів. Однак він не підтримує гнучких налаштувань або роботи з іншими типами акумуляторів.

3. Контролер ATmega328: Гнучкість і функціональність

Контролер ATmega328 — це мікроконтролер із архітектурою AVR, який широко застосовується у проєктах, що вимагають гнучкості та програмованості. На відміну від інтегрованих контролерів, таких як BQ24133 чи TP4056, ATmega328 дозволяє створювати зарядні пристрої з адаптивними алгоритмами заряджання, які враховують специфічні особливості різних типів акумуляторів.

Основні переваги ATmega328 у зарядних пристроях:

1. Гнучкість програмування:
   На відміну від інтегральних контролерів, алгоритми заряджання на ATmega328 можна змінювати та налаштовувати під специфічні потреби. Наприклад, можна реалізувати режими заряджання для Li-ion, NiMH або свинцево-кислотних акумуляторів із врахуванням їхніх унікальних характеристик.
2. Можливість інтеграції додаткових функцій:
   Завдяки доступу до портів введення/виведення ATmega328 може керувати додатковими компонентами, такими як датчики температури, дисплеї або засоби віддаленого контролю (UART, I2C, SPI). Це дозволяє створювати "розумні" зарядні пристрої з розширеними функціями моніторингу.
3. Підтримка адаптивного заряджання:
   ATmega328 може забезпечити покрокове заряджання, аналогічне до BQ24133:
   - Передзарядка: алгоритм визначає сильно розряджені акумулятори та обмежує струм для безпечного відновлення.
   - Швидке заряджання: програмне забезпечення регулює струм до певного рівня напруги, враховуючи тип батареї.
   - Стабілізація: забезпечується контроль напруги до повного заряджання без ризику перенапруги.

4. Інтеграція засобів захисту:
   Як і в TP4056, в системах із ATmega328 можна реалізувати захист від:
   - Перевантаження по струму.
   - Перегріву.
   - Надмірного розряджання чи перезаряджання.
5. Доступність та економічність:
   Незважаючи на ширші можливості, ATmega328 залишається доступним і економічно вигідним рішенням, особливо в проєктах, що вимагають кастомізації.

Недоліки ATmega328 порівняно з інтегрованими контролерами:

- Складність розробки: Для використання ATmega328 потрібно розробляти прошивку з нуля, що вимагає певного рівня знань у програмуванні.
- Використання зовнішніх компонентів: Для реалізації зарядного пристрою на основі ATmega328 потрібно підключати додаткові компоненти, такі як MOSFET-транзистори, АЦП для вимірювання напруги та струму, а також джерело стабільної напруги.

Висновок

Мікроконтролер ATmega328 є ідеальним вибором для створення зарядних пристроїв із розширеним функціоналом, який можна адаптувати до специфічних вимог. Хоча він вимагає більше зусиль для розробки, порівняно з інтегральними контролерами, його переваги у гнучкості та розширюваності роблять його оптимальним вибором для універсальних систем заряджання[18,19].

Таблиця 1.2 Порівння процесорів

Порівняльна характеристика

Характеристика	BQ24133	TP4056	ATmega328P
Вартість	Середня	Низька	Низька
Гнучкість налаштувань	Обмежена	Відсутня	Висока
Підтримка типів батарей	Li-ion	Li-ion	Універсальна
Функції захисту	Вбудовані	Вбудовані	Програмовані

Опис і порівняння різновидів плат Arduino

Arduino — це платформа для розробки вбудованих систем, що складається з апаратної (плати з мікроконтролером) та програмної (Arduino IDE) частин. Arduino випускає різноманітні плати, які відрізняються мікроконтролерами, функціоналом, розмірами та вартістю. У цьому розділі розглянемо найпоширеніші плати Arduino, включаючи ті, що базуються на мікроконтролері ATmega328.

1. Arduino Uno (ATmega328P)(*Рис. 1.1*)

Опис:

- Arduino Uno — це одна з найпопулярніших і універсальних плат для початківців.
- Оснащена мікроконтролером ATmega328P.
- Має 14 цифрових входів/виходів (6 із них підтримують ШІМ) і 6 аналогових входів.

Характеристики:

- Пам'ять: 32 КБ Flash, 2 КБ SRAM, 1 КБ EEPROM.
- Тактова частота: 16 МГц.
- Живлення: 5 В (USB) або 7–12 В (через роз'єм живлення).
- Інтерфейси: UART, SPI, I2C.

Переваги:

- Простота використання, наявність великої спільноти.
- Підходить для більшості типових проєктів.
- Доступність і низька вартість.

Недоліки:

- Обмежена кількість входів/виходів для складних проєктів.
- Відсутність бездротової комунікації.

Рис. 1.1 Arduino Uno (ATmega328P)

2. Arduino Nano (ATmega328P)

Опис:

- Компактна версія Arduino Uno.
- Оснащена тим самим мікроконтролером ATmega328P.
- Має всі функції Uno, але в меншому розмірі.

Характеристики:

- Розмір: 18 x 45 мм.
- Живлення: 5 В (через USB) або 7–12 В (через VIN).
- Розширення: доступно 22 входи/виходи.

Переваги:

- Компактність, підходить для вбудованих систем.
- Повністю сумісна з кодом і бібліотеками Arduino Uno.

Недоліки:

- Менша зручність для підключення компонентів через паяні контакти.

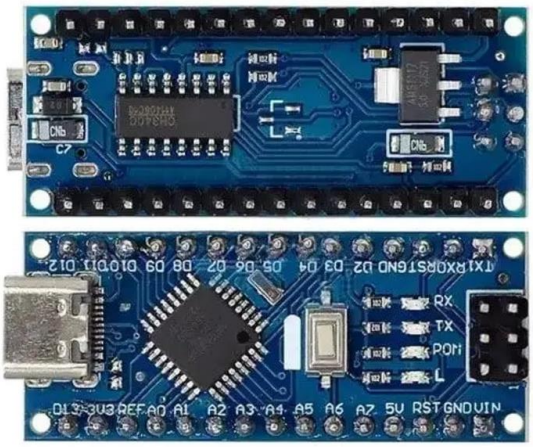


Рис. 1.2 Arduino Nano (ATmega328P)

3. Arduino Mega (ATmega2560)(Рис.*1.3*)

Опис:

- Плата для складних проєктів, які потребують великої кількості входів/виходів.
- Використовує мікроконтролер ATmega2560.

Характеристики:

- Пам'ять: 256 КБ Flash, 8 КБ SRAM, 4 КБ EEPROM.
- 54 цифрових входи/виходи (15 із них підтримують ШІМ), 16 аналогових входів.
- Тактова частота: 16 МГц.

Переваги:

- Велика кількість входів/виходів.
- Підходить для робототехніки, великих систем.

Недоліки:

- Вища вартість і розмір.
- Не завжди виправдана для простих проєктів.

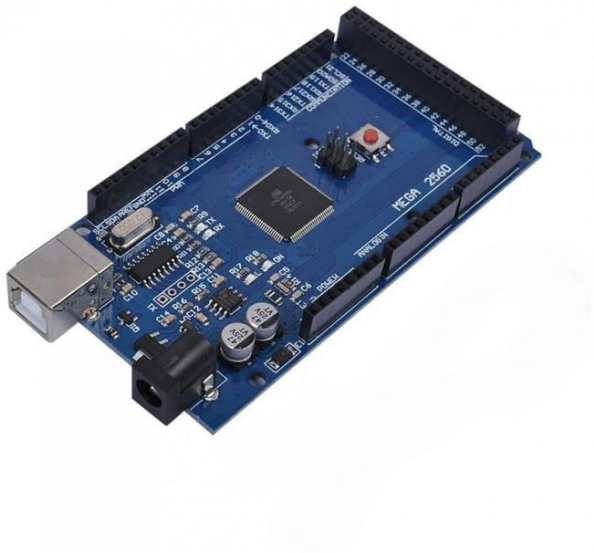


Рис.1.3 Arduino Mega (ATmega2560)

4. Arduino Leonardo (ATmega32U4)(*Рис. 1.4*)

Опис:

- Особливість плати — мікроконтролер ATmega32U4 із вбудованою підтримкою USB.
- Може імітувати пристрої введення, такі як клавіатура чи миша.

Характеристики:

- Пам'ять: 32 КБ Flash, 2.5 КБ SRAM, 1 КБ EEPROM.
- 20 цифрових входів/виходів (7 із них підтримують ШІМ), 12 аналогових входів.
- Тактова частота: 16 МГц.

Переваги:

- Вбудована підтримка USB для емуляції HID-пристроїв.
- Підходить для інтерактивних проєктів.

Недоліки:

- Менша популярність порівняно з Uno.
- Менша підтримка з боку спільноти.

Рис. 1.4 Arduino Leonardo (ATmega32U4)

5. Arduino Due (ARM Cortex-M3)(*Рис. 1.5*)

Опис:

- Плата для високопродуктивних систем із 32-бітним процесором ARM Cortex-M3.

Характеристики:

- Пам'ять: 512 КБ Flash, 96 КБ SRAM.
- Тактова частота: 84 МГц.
- Живлення: 3.3 В.

Переваги:

- Висока продуктивність.
- Більша кількість входів/виходів: 54 цифрових, 12 аналогових.

Недоліки:

- Несумісність із бібліотеками, розробленими для 8-бітних платформ.
- Працює на 3.3 В, що може вимагати використання логічних рівнів для зовнішніх компонентів.

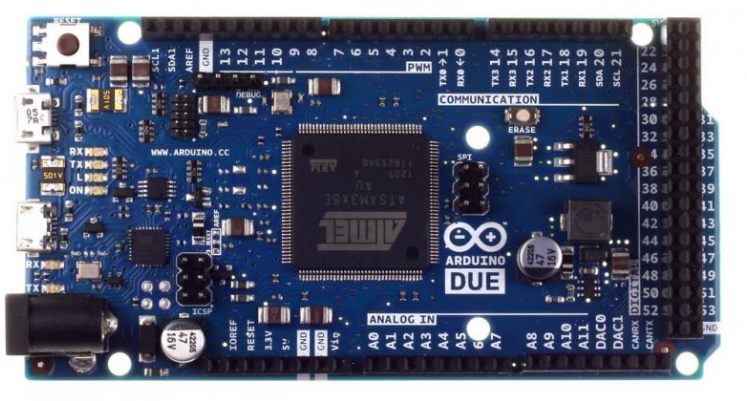


Рис. 1.5 Arduino Due (ARM Cortex-M3)

6. Arduino Pro Mini ATmega328P (*Рис. 1.6*)

Опис:

- Надзвичайно компактна версія Arduino без роз'ємів для розробників із досвідом.

Характеристики:

- Мікроконтролер: ATmega328P.
- Живлення: 3.3 В або 5 В.

Переваги:

- Дуже компактний розмір.
- Підходить для фінальних пристроїв.

Недоліки:

- Не має вбудованого USB-інтерфейсу (потрібен програматор).

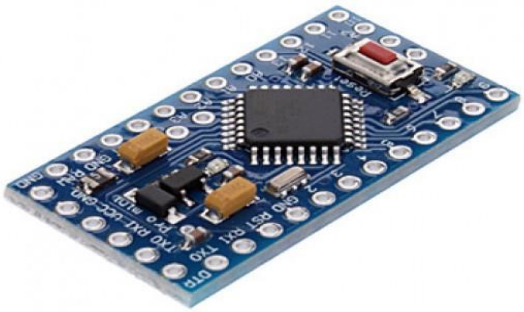


Рис. 1.6 Arduino Pro Mini ATmega328P

Таблиця1.2 Порівняння плат Arduino

Порівняльна таблиця

Модель	Мікроконтролер	Пам'ять (Flash/SRAM)	Входи/ виходи	Тактова частота	Особливості
Arduino Uno	ATmega328P	32 КБ / 2 КБ	14 цифрових, 6 аналогових	16 МГц	Стандарт для початківців
Arduino Nano	ATmega328P	32 КБ / 2 КБ	22 (цифрові/ аналогові)	16 МГц	Компактний розмір
Arduino Mega	ATmega2560	256 КБ / 8 КБ	54 цифрових, 16 аналогових	16 МГц	Для масштабних проєктів
Arduino Leonardo	ATmega32U4	32 КБ / 2.5 КБ	20 цифрових, 12 аналогових	16 МГц	Підтримка HID-пристроїв
Arduino Due	ARM Cortex-M3	512 КБ / 96 КБ	54 цифрових, 12 аналогових	84 МГц	Висока продуктивність
Arduino Pro Mini	ATmega328P	32 КБ / 2 КБ	14 цифрових, 6 аналогових	16 МГц	Мінімалістичний, для вбудованих систем

Arduino Nano чудово підходить для зарядного пристрою

1. Гнучкість в адаптації алгоритмів заряджання:

   Завдяки програмованості, можна реалізувати різні режими заряджання для різних типів акумуляторів, таких як Li-ion, NiMH, свинцево-кислотні тощо. Наприклад:

   - Передзарядка для сильно розряджених акумуляторів.
   - Швидке заряджання при постійному струмі.
   - Завершальна стабілізація при постійній напрузі.

2. Інтеграція датчиків:

   Arduino Nano може працювати з датчиками температури, струму, напруги, що забезпечує безпечне заряджання із захистом від:

   - Перевантаження по струму.
   - Перегріву.
   - Перенапруги.

3. Можливість реалізації інтерфейсу для користувача:

   За допомогою OLED-дисплеїв або LCD-екранів можна виводити поточні параметри заряджання, такі як напруга, струм, залишковий час до повного заряджання.

4. Підтримка віддаленого моніторингу:

   Завдяки UART або I2C інтерфейсам, Arduino Nano дозволяє підключати модулі бездротового зв'язку, такі як Bluetooth чи Wi-Fi, для дистанційного контролю процесу заряджання.

Низька вартість та доступність: Arduino Nano є економічним рішенням, що дозволяє створити функціональний зарядний пристрій без значних фінансових витрат.

Висновок

Arduino Nano ідеально підходить для створення зарядного пристрою завдяки своїм компактним розмірам, простоті програмування, гнучкості у підключенні периферії та низькій вартості. Це дозволяє розробити надійний та ефективний зарядний пристрій, адаптований до сучасних вимог енергоефективності, безпеки та зручності використання.

2.2 Вибір апаратної платформи (опис та можливості)

Універсальні зарядні пристрої

Універсальні зарядні пристрої здатні працювати з різними типами акумуляторів, такими як NiMH, Li-ion, LiFePO4, SLA. Такі пристрої зазвичай базуються на мікроконтролерах або програмованих інтегральних схемах, які забезпечують складні алгоритми заряджання, контроль параметрів батареї та захисні функції.

Прикладом таких пристроїв є багатофункціональні зарядні станції для електромобілів або лабораторні зарядні пристрої, які дозволяють налаштовувати параметри заряджання для різних батарей через інтерфейси USB або Bluetooth. Недоліками універсальних пристроїв є їхня вища вартість і складність конструкції.

Використання мікроконтролерів у зарядних пристроях

Мікроконтролери, такі як ATmega328P, відкривають нові можливості у проєктуванні зарядних пристроїв:

- Гнучке програмування: Завдяки підтримці мов програмування (C, Arduino IDE) можна створювати унікальні алгоритми заряджання для різних типів акумуляторів.
- Реалізація захисних функцій: Мікроконтролер дозволяє додавати контроль температури, захист від перевантаження, перевищення напруги та інші функції.
- Підтримка інтерфейсів: ATmega328P підтримує UART, SPI, I2C, що дозволяє реалізувати віддалений контроль через додаткові модулі, такі як Bluetooth або Wi-Fi.

У розробці програмованого зарядного пристрою важливим етапом є вибір апаратної платформи, яка забезпечує необхідну продуктивність, функціональність і гнучкість для реалізації завдань. Для цієї розробки обрано платформу Arduino на основі мікроконтролера ATmega328P. Її вибір обумовлений низкою переваг, серед яких простота використання, доступність, розвинена екосистема та підтримка широкого спектра бібліотек.

Arduino — це інтегрована апаратно-програмна платформа, призначена для проектування електронних пристроїв, яка забезпечує тісну взаємодію з навколишнім фізичним середовищем. Вона відрізняється від традиційних персональних комп'ютерів, які функціонують переважно у віртуальному просторі, не маючи можливості безпосередньо взаємодіяти з реальним світом[20].

Пристрої, створені на базі платформи Arduino, здатні приймати сигнали від різноманітних цифрових та аналогових датчиків, а також управляти різними виконавчими механізмами, такими як мотори, світлодіоди та реле. Arduino є готовою платформою, яка поєднує в собі апаратні та програмні компоненти. Основними елементами цієї системи є плата-контролер, що виконує функції введення/виведення, а також середовище розробки, яке базується на мовах програмування Processing і Wiring.

Проекти на платформі Arduino можуть функціонувати автономно, а також інтегруватися з програмами, що працюють на персональних комп'ютерах, такими як Adobe Flash, Processing або MaxMSP. Це дозволяє реалізувати складні системи, які поєднують фізичні елементи з програмним забезпеченням.

Існує безліч моделей Arduino, кожна з яких розроблена для специфічних задач і має свої особливості. Проте більшість із них поділяють спільні компоненти, такі як:

- Роз'єм живлення: Усі плати Arduino мають можливість підключення до джерела живлення. Наприклад, плата Arduino Uno може живитися як від USB-кабелю, що підключається до персонального комп'ютера, так і від окремого адаптера, підключеного до спеціального роз'єму на платі. Рекомендована напруга живлення для Arduino варіюється від 6 до 12 вольт. Крім того, USB використовується для завантаження програмного забезпечення (скетчів) на плату.

Arduino стає все більш популярним серед розробників, інженерів та любителів, адже дозволяє легко реалізувати проекти з використанням електроніки та автоматизації. Завдяки відкритій архітектурі та великій кількості доступних бібліотек, платформа підтримує широкий спектр застосувань — від простих до складних систем, що взаємодіють з навколишнім середовищем.

Характеристики платформи Arduino ATmega328P

Для реалізації системи управління зарядом акумуляторних батарей у даному проєкті використовується мікроконтролер Arduino на базі Atmega 328. Це потужний, гнучкий і широко застосовуваний контролер, який дозволяє легко інтегрувати різноманітні функції управління зарядним процесом, збирати та обробляти дані з датчиків, а також реалізовувати різні алгоритми заряду для різних типів акумуляторних батарей.

Мікроконтролер Arduino Atmega 328 є ключовим елементом системи, який поєднує всі апаратні компоненти та забезпечує надійну роботу зарядного пристрою.

Мікроконтролер Atmega 328 є серцем платформи Arduino Uno і має такі основні технічні характеристики:

1. Мікроконтролер: ATmega328P
2. Тактова частота: 16 МГц
3. Обсяг флеш-пам’яті: 32 КБ (2 КБ зайняті завантажувачем)
4. Обсяг оперативної пам’яті (SRAM): 2 КБ
5. EEPROM: 1 КБ для зберігання постійних даних
6. Кількість цифрових входів/виходів: 14 (6 з яких можуть використовуватися як ШІМ-канали)
7. Аналогові входи: 6 (10-бітний АЦП)
8. Інтерфейси: UART, SPI, I^2C
9. Живлення: від 5 В постійного струму через USB або зовнішній джерело (7-12 В)
10. Низьке енергоспоживання: підтримка режимів енергозбереження для економії ресурсу.

Мікроконтролер Atmega 328 дозволяє реалізувати всі необхідні функції зарядного пристрою, що потребуються для управління процесом заряду акумуляторних батарей. Ось основні аспекти використання цього контролера у проєкті:

1. Збір даних від датчиків: Мікроконтролер підключається до датчиків напруги та струму, таких як ACS712 або INA219, що дозволяють вимірювати поточні параметри зарядного процесу. Завдяки вбудованому аналогово-цифровому перетворювачу (АЦП) Atmega 328 здатен зчитувати аналогові сигнали з високою точністю та перетворювати їх на цифрові значення для подальшої обробки.
2. Реалізація алгоритмів заряду: Atmega 328 дозволяє програмувати різні алгоритми заряду, такі як постійний струм, постійна напруга, імпульсний заряд, а також контроль температури батареї. Програмне забезпечення на базі цього мікроконтролера здатне автоматично перемикатися між алгоритмами, залежно від стану АКБ. Наприклад, після досягнення заданої напруги заряд переходить у режим підтримки або підзарядки.

3. Контроль температури і безпеки: Arduino Atmega 328 може обробляти дані від термодатчиків (наприклад, термісторів або цифрових сенсорів DS18B20) для запобігання перегріву батареї. Якщо температура перевищує встановлені межі, мікроконтролер автоматично припиняє процес заряду або знижує струм, що запобігає потенційній небезпеці.
4. Виведення інформації на дисплей: За допомогою Arduino можна керувати рідкокристалічним дисплеєм (LCD) для відображення основних параметрів заряду, таких як напруга, струм, час до завершення заряду, стан акумулятора та інші. Це забезпечує зручний інтерфейс для користувача.
5. Керування режимами заряду: Мікроконтролер використовує енкодер і кнопки для вибору та налаштування режимів заряду. Це дозволяє користувачеві задавати необхідні параметри заряду для конкретного типу батареї або вибирати автоматичний режим, де мікроконтролер самостійно підбирає оптимальні налаштування.
6. Автоматизація процесу: Atmega 328 дозволяє повністю автоматизувати процес заряду, мінімізуючи втручання людини. Мікроконтролер може контролювати параметри АКБ, обирати відповідний алгоритм заряду та коригувати струм і напругу у реальному часі. Це дозволяє забезпечити ефективний та безпечний процес заряду без ризику перезаряду або пошкодження батареї.
7. Збереження даних: Завдяки EEPROM пам'яті, мікроконтролер здатен зберігати дані про стан АКБ та параметри заряду навіть після відключення живлення. Це може бути корисним для проведення аналізу стану акумуляторів і для виконання контролю тренувальних циклів.

Arduino Nano — це компактна, потужна та зручна в роботі плата на базі мікроконтролера ATmega328, яка ідеально підходить для проєктів, що потребують високої функціональності у невеликому форм-факторі. Для розробки зарядного пристрою саме Arduino Nano було обрано завдяки її численним перевагам:

Компактні розміри: Arduino Nano має розміри всього 18x45 мм, що дозволяє інтегрувати її навіть у

невеликі корпуси зарядних пристроїв, зберігаючи простір для інших компонентів.

1. Простота у використанні:
   Arduino Nano легко програмується завдяки відкритій платформі Arduino IDE, що спрощує створення складних алгоритмів заряджання без глибокого занурення в низькорівневе програмування.
2. Доступність GPIO-портів:
   Плата забезпечує 14 цифрових і 8 аналогових входів/виходів, що дозволяє підключати широкий спектр периферійних компонентів, таких як:
   - Датчики напруги та струму.
   - Дисплеї для виведення інформації.
   - Інтерфейси зв’язку (UART, SPI, I2C) для зовнішніх модулів.
3. Вбудовані можливості АЦП:
   Завдяки 10-бітному аналого-цифровому перетворювачу (АЦП) ATmega328, Arduino Nano дозволяє точно вимірювати напругу та струм, що критично важливо для контролю процесу заряджання.
4. Підтримка живлення від USB та зовнішніх джерел:
   Arduino Nano підтримує живлення від USB (5 В) або зовнішніх джерел (6–12 В), що забезпечує гнучкість при розробці портативних або стаціонарних пристроїв.
5. Можливість оновлення прошивки:
   Arduino Nano дозволяє швидко оновлювати алгоритми заряджання шляхом перепрошивки через USB, що є важливим для тестування та покращення функціоналу.

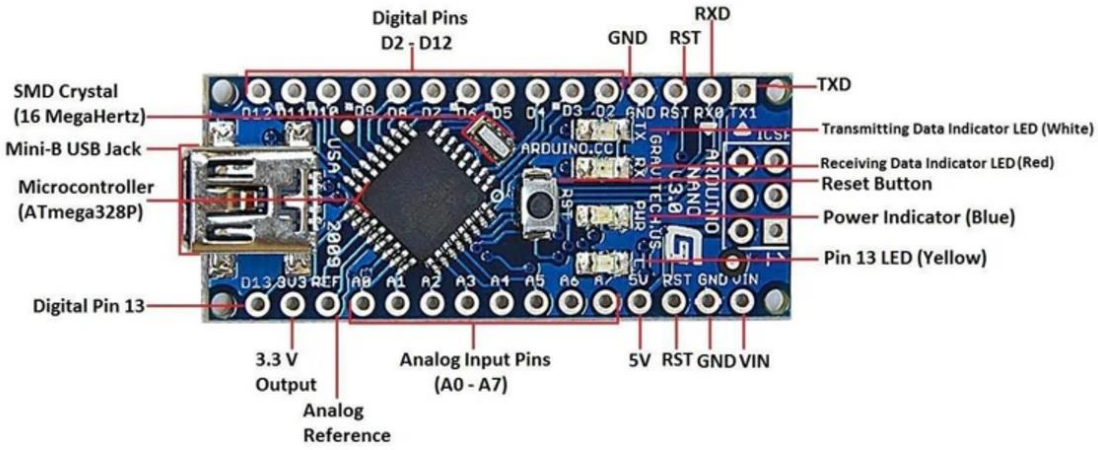
Гнучкість в адаптації алгоритмів заряджання:
Завдяки програмованості, можна реалізувати різні режими заряджання для різних типів акумуляторів, таких як Li-ion, NiMH, свинцево-кислотні тощо. Наприклад:

- Передзарядка для сильно розряджених акумуляторів.
- Швидке заряджання при постійному струмі.
- Завершальна стабілізація при постійній напрузі.

8. Інтеграція датчиків:
   Arduino Nano може працювати з датчиками температури, струму, напруги, що забезпечує безпечне заряджання із захистом від:
   - Перевантаження по струму.
   - Перегріву.
   - Перенапруги.
9. Можливість реалізації інтерфейсу для користувача:
   За допомогою OLED-дисплеїв або LCD-екранів можна виводити поточні параметри заряджання, такі як напруга, струм, залишковий час до повного заряджання.
10. Підтримка віддаленого моніторингу:
    Завдяки UART або I2C інтерфейсам, Arduino Nano дозволяє підключати модулі бездротового зв’язку, такі як Bluetooth чи Wi-Fi, для дистанційного контролю процесу заряджання.
11. Низька вартість та доступність:
    Arduino Nano є економічним рішенням, що дозволяє створити функціональний зарядний пристрій без значних фінансових витрат.

Висновок

Arduino Nano ідеально підходить для створення зарядного пристрою завдяки своїм компактним розмірам, простоті програмування, гнучкості у підключенні периферії та низькій вартості. Це дозволяє розробити надійний та ефективний зарядний пристрій, адаптований до сучасних вимог енергоефективності, безпеки та зручності використання.(Рис. 1.7)



Arduino Nano V3.0 Pinout

Рис. 1.7 Arduino Nano V3.0 Pinout

2.3 Особливості роботи з обраною платформою (програмні та апаратні аспекти)

Апаратні аспекти роботи з Arduino Nano

1. Компактний форм-фактор

   Arduino Nano є однією з найменших плат у лінійці Arduino, що дозволяє інтегрувати її у компактні зарядні пристрої. Її розміри (18x45 мм) роблять можливим розробку портативних рішень.

2. Процесор ATmega328P

   У центрі платформи — мікроконтролер ATmega328P з тактовою частотою 16 МГц. Цей контролер забезпечує необхідну обчислювальну потужність для виконання алгоритмів заряджання, обробки даних від сенсорів і комунікації з периферійними модулями.

3. Енергоспоживання

   Arduino Nano підтримує живлення від USB (5 В) або зовнішнього джерела (6–12 В), що дозволяє використовувати її у зарядних пристроях із різними джерелами живлення. Для енергоефективних застосунків доступні режими зниження енергоспоживання, які можна реалізувати програмно.

4. Інтеграція периферії

   Arduino Nano має 14 цифрових входів/виходів (6 з яких підтримують ШІМ)

і 8 аналогових входів. Ця кількість дозволяє підключати сенсори (напруги, струму, температури), дисплеї, кнопки керування та інші компоненти.

5. Інтерфейси зв’язку

   Мікроконтролер підтримує кілька типів комунікацій:

   - UART для роботи з модулями Bluetooth або послідовного моніторингу.
   - SPI для підключення високошвидкісних датчиків або пам’яті.
   - I²C для зв’язку з дисплеями чи іншими мікроконтролерами.

6. Вбудований USB-конвертер

   Плата обладнана чіпом CH340 або FT232 для прямого підключення до комп’ютера через USB, що спрощує програмування та налагодження.

Програмні аспекти роботи з Arduino Nano

1. Arduino IDE

   Офіційне середовище розробки Arduino IDE забезпечує зручний інтерфейс для написання, компіляції та завантаження програм у плату. IDE підтримує мову програмування, засновану на C/C++, та має бібліотеки для роботи з периферією.

2. Гнучкість програмування

   Arduino Nano дозволяє створювати власні алгоритми заряджання для різних типів акумуляторів. Наприклад:

   - Логіка контролю етапів заряджання (передзарядка, швидке заряджання, стабілізація).
   - Захист від перенапруги та перегріву, програмовані обмеження струму.

3. Бібліотеки для периферії

   Arduino має численні готові бібліотеки для роботи з:

   - Дисплеями (LCD, OLED).
   - Датчиками (температури, струму, напруги).
   - Інтерфейсами зв’язку (UART, I²C, SPI).

   Це скорочує час розробки та мінімізує помилки у коді.

4. Підтримка оновлень та налагодження
   Arduino Nano дозволяє легко оновлювати програмне забезпечення через USB. Для моніторингу та налагодження можна використовувати послідовний монітор Arduino IDE.
5. Реалізація багатозадачності
   Завдяки використанню таймерів і переривань, Arduino Nano може виконувати кілька завдань одночасно, наприклад:
   - Контроль температури акумулятора.
   - Управління режимами заряджання.
   - Обробка введення користувача (наприклад, через кнопки чи енкодери).

   Переваги роботи з Arduino Nano

1. Доступність та активна спільнота
   Arduino Nano має велику базу користувачів і документації. Це дозволяє легко знайти готові приклади та рішення для конкретних завдань, таких як контроль процесу заряджання.
2. Гнучкість та масштабованість
   Платформа дозволяє реалізувати широкий спектр функцій, від простого заряджання до складних систем із віддаленим моніторингом і керуванням.
3. Низька вартість
   Arduino Nano — це економічно вигідне рішення для створення універсальних зарядних пристроїв без значного збільшення вартості проєкту.

   Недоліки та обмеження

1. Обмежена обчислювальна потужність
   Arduino Nano не підходить для дуже складних обчислень або обробки великих обсягів даних у реальному часі. Для таких задач краще використовувати потужніші платформи, наприклад, Raspberry Pi або STM32.
2. Обмежена кількість пам’яті
   ATmega328 має 32 КБ флеш-пам’яті для коду та 2 КБ оперативної пам’яті, що може стати обмеженням для великих програм.

Висновок

Arduino Nano — це ідеальний вибір для реалізації зарядного пристрою завдяки простоті програмування, компактності, гнучкості у роботі з периферією та підтримці численних бібліотек. Вона дозволяє розробляти надійні та функціональні рішення для заряджання різних типів акумуляторів, враховуючи потреби безпеки, енергоефективності та користувацької зручності[2].

РОЗДІЛ 3. РОЗРОБКА ПРИСТРОЮ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОКОНТРОЛЕРА

Пристрій оснащено рідкокристалічним дисплеєм з роздільною здатністю 162 символів, на якому відображаються поточні параметри заряду (напруга, струм, ємність) та інформаційні повідомлення.

Контроль заряду здійснюється через енкодер та кнопки для вибору параметрів.

Зарядний пристрій реалізує кілька алгоритмів заряду:

1. Попередній заряд для сильно розряджених акумуляторів.

2. Основний заряд за комбінованим методом: стабілізація струму на початковому етапі і стабілізація напруги на завершальному.

3. Режим "качелі" в кінці основного заряду для балансування елементів.

4. Дозаряд для максимального заповнення ємності.

5. Режим зберігання для підтримання заряду під час довготривалого невикористання акумулятора.

Пристрій також забезпечує розрядний режим, вимірювання внутрішнього опору акумулятора і проведення контрольно-тренувальних циклів (КТЦ), що допомагають підтримувати акумулятор у гарному стані.

Алгоритм зарядки малими струмами за методикою Браніміра дозволяє більш ефективно заряджати акумулятори, збільшуючи їх довговічність.

Зарядний пристрій постійно контролює напругу та струм зарядки/розрядки, автоматично обчислюючи і зберігаючи в пам'ять дані про кількість заряджених ампер-годин та ват-годин, що забезпечує повний контроль над процесом зарядки.

Цей зарядний пристрій призначений для забезпечення безпечного та ефективного процесу заряджання, автоматизації контролю та запобігання пошкодженню акумуляторів завдяки інтелектуальним алгоритмам.

Основним завданням цього проекту є створення системи управління зарядом акумуляторних батарей (АКБ) на базі мікроконтролера (МК). Ця система

включає як апаратні компоненти (блок живлення, мікроконтролер, датчики), так і програмне забезпечення, яке керуватиме процесом заряду.

Розроблений пристрій заряджання складається з таких основних модулів(Додаток Д,Е):

1. Блок живлення: Першим кроком є створення блоку живлення, що забезпечує перетворення мережевої напруги (220-230 В змінного струму) в стабільну напругу постійного струму, яка необхідна для живлення мікроконтролера, дисплея та датчиків. Блок живлення повинен відповідати вимогам стабільності та безпеки.

2. Мікроконтролер (МК): Мікроконтролер є центральним елементом пристрою, який забезпечує збір, обробку даних та керування процесом заряду. МК обирає режим заряду в залежності від параметрів акумулятора, контролює поточну напругу та струм заряду, а також змінює алгоритми заряду на різних етапах.

3. Система управління струмом заряду: Система управління струмом включає схему стабілізації струму, яка дозволяє мікроконтролеру регулювати величину струму, що подається на АКБ. Це критично важливо для запобігання перезаряду або перегріву акумулятора, що може призвести до зниження його ефективності та пошкодження.

4. Датчики струму і напруги: Для точного регулювання процесу заряду мікроконтролер отримує інформацію від датчиків струму і напруги. Ці датчики дозволяють відслідковувати поточні параметри АКБ та подавати зворотний зв’язок для коригування режимів заряду. Це забезпечує точний контроль процесу та запобігає перезаряду або недозаряду акумулятора.

5. Дисплей для виведення інформації: Рідкокристалічний дисплей відображає всю необхідну інформацію про поточний процес заряду, включаючи напругу, струм, залишковий час до завершення заряду та стан акумулятора. Це дозволяє користувачу в реальному часі моніторити процес та своєчасно втручатися у разі необхідності.

6. Енкодер для керування: Енкодер використовується для налаштування параметрів заряду та вибору режимів роботи пристрою. Він забезпечує зручну

взаємодію з користувачем, дозволяючи змінювати режими заряду та налаштування МК без складних технічних процедур. Енкодер є ключовим компонентом для управління та налаштування параметрів в системах з мікроконтролерами, таких як Arduino. В даному проєкті, де мікроконтролер Arduino на базі Atmega 328 використовується для управління процесом заряду акумуляторних батарей, енкодер виконує функцію інтерфейсу для введення даних та вибору режимів роботи пристрою.

Основні функції енкодера в проєкті:

1) Регулювання параметрів заряду.

Енкодер використовується для зміни налаштувань зарядного пристрою, таких як струм заряду, напруга, тип акумулятора та інші параметри. Повертаючи енкодер, користувач може збільшувати або зменшувати значення цих параметрів, що робить процес налаштування простим та інтуїтивним.

2) Вибір режимів роботи:

Енкодер також забезпечує можливість вибору різних режимів роботи зарядного пристрою, таких як основний заряд, розряд, контрольний цикл або підтримка заряду. Це дозволяє користувачу легко переходити між режимами, адаптуючи пристрій до поточного стану акумулятора.

3) Кнопка енкодера для підтвердження вибору:

Більшість енкодерів для Arduino мають вбудовану кнопку. У цьому проєкті кнопка енкодера використовується для підтвердження вибраних параметрів або режимів роботи після їх налаштування. Це забезпечує зручний спосіб взаємодії з системою без необхідності використовувати додаткові кнопки.

Енкодер, що використовується в нашому проєкті, є обертальним (ротаційним) енкодером, який зазвичай застосовується для кодування кута обертання в електронних системах. Він генерує імпульси на виходах при обертанні, які Arduino може зчитувати для визначення напрямку та кількості обертів.

Енкодер має два основні виходи:

- A та B: це дві фази, які зсуваються одна відносно одної. Arduino визначає напрямок обертання на основі того, яка з фаз змінюється першою.

- Кнопка: для підтвердження вибору або виклику певних функцій.

Переваги використання енкодера:

• Точність налаштувань: енкодер дозволяє дуже точно налаштовувати параметри заряду за рахунок можливості вибору навіть найменших кроків при обертанні.

• Зручність у користуванні: простота у використанні (*Рис. 2.8*)

Рис. 2.8 Енкодер

Дисплей LCD 1602

Дисплей LCD 1602 I2C є важливим елементом інтерфейсу користувача в проекті програмованого зарядного пристрою для Arduino. Він дозволяє відображати інформацію про процес заряду, налаштування параметрів, а також різноманітні повідомлення, що забезпечує зручність та зрозумілість взаємодії з пристроєм.

Основні характеристики дисплея LCD 1602 I2C:

1) Розміри: Дисплей має 16 колонок та 2 рядки, що дозволяє відображати до 32 символів одночасно.

2) I2C Інтерфейс: З використанням I2C (Inter-Integrated Circuit) інтерфейсу, LCD 1602 може підключатися до Arduino за допомогою лише двох проводів (SDA та SCL), що значно спрощує підключення та зменшує кількість необхідних пінів на мікроконтролері.

3) Підсвітка: Багато моделей LCD 1602 мають підсвітку, що покращує видимість інформації на екрані в умовах недостатнього освітлення.

Використання дисплея у проекті:

4) Відображення параметрів заряду.

Дисплей використовується для візуалізації поточних параметрів заряду, таких як напруга на акумуляторі, струм заряду, час зарядки, статус режиму роботи (основний заряд, підзарядка тощо). Це дозволяє користувачу легко відстежувати процес заряду та вчасно реагувати на зміни.

1. Налаштування параметрів.

Завдяки інтерактивності з енкодером, дисплей показує доступні параметри для зміни, що дозволяє користувачу легко налаштовувати такі величини, як максимальний струм заряду, тип акумулятора, режими роботи тощо.

2. Виведення інформаційних повідомлень.

Дисплей може показувати різноманітні інформаційні та попереджувальні повідомлення, такі як: повідомлення про закінчення заряду; попередження про перевантаження або перегрів; нагадування про перевірку акумулятора.

3. Програмне забезпечення для дисплея.

Для роботи з дисплеєм LCD 1602 I2C в проекті використовується бібліотека LiquidCrystal_I2C, яка спрощує управління дисплеєм. Ця бібліотека дозволяє легко ініціалізувати дисплей, виводити текст, а також керувати підсвіткою. Основні команди:

- Ініціалізація дисплея:

```
cpp
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Адреса I2C, кількість колонок та рядків
lcd.begin();
```

- Виведення тексту:

```
cpp
lcd.setCursor(0, 0);  // Встановлення курсора на першу колонку, перший рядок
lcd.print("Струм: 1A");
```

- Оновлення інформації: В процесі роботи зарядного пристрою, дані на дисплеї постійно оновлюються, щоб відображати актуальні значення (Рис. 2.9).

Рис. 2.9 LCD дисплей 1602 I2C

Висновок

Дисплей LCD 1602 I2C є незамінним компонентом проекту програмованого зарядного пристрою для Arduino. Він забезпечує зручний і зрозумілий інтерфейс для користувача, дозволяючи ефективно контролювати та налаштовувати процес заряду акумуляторних батарей. Завдяки простоті підключення та управління, LCD 1602 I2C робить проект більш доступним і функціональним (Рис. 2.9)

Датчик температури DS18B20

Датчик температури DS18B20 — це цифровий сенсор, призначений для вимірювання температури в широкому діапазоні. Він часто використовується в проектах на основі мікроконтролерів, таких як Arduino, завдяки своїй точності, простоті підключення та можливості використання одного проводу (1-Wire) для передачі даних.

Основні характеристики DS18B20

1. Цифровий вихід:
    - DS18B20 має цифровий вихід, що дозволяє отримувати температурні показники без необхідності аналогово-цифрового перетворення. Це знижує ризик похибок, пов'язаних з аналоговими сигналами.
2. Діапазон вимірювань:

- Датчик може вимірювати температуру в діапазоні від -55°C до +125°C з точністю до ±0.5°C в діапазоні від -10°C до +85°C.

3. Роздільна здатність:
   - DS18B20 підтримує вибір роздільної здатності вимірювання: 9, 10, 11 або 12 біт. Це дозволяє налаштовувати точність в залежності від потреб проекту.
4. Інтерфейс 1-Wire:
   - DS18B20 використовує протокол 1-Wire для зв'язку з мікроконтролерами, що означає, що до одного піну Arduino можна підключити кілька датчиків, спрощуючи проводку та зменшуючи кількість пінів, які необхідно використовувати.
5. Живлення:
   - Датчик може живитися як від зовнішнього джерела живлення (3.0V - 5.5V), так і в режимі "паралельного живлення", використовуючи енергію з лінії даних, що також спрощує його інтеграцію в системи (Рис. 2.10).

Рис. 2.10. Датчик температури DS18B20

Використання DS18B20 у проекті

1. Підключення:
   - Для підключення датчика до Arduino зазвичай використовують один пін для даних, один пін для живлення та один пін для заземлення. На лінії даних зазвичай підключають резистор (4.7kΩ) для підтягування.

2. Програмне забезпечення:
    - Для роботи з датчиком DS18B20 в проектах на Arduino використовують бібліотеки, такі як OneWire і DallasTemperature. Ці бібліотеки спрощують обробку даних і роблять взаємодію з датчиком зручною.

Основні команди:

- Ініціалізація бібліотек:

cpp

```
#include <OneWire.h>
#include <DallasTemperature.h>
OneWire oneWire(pin);  // pin — це пін, до якого підключений датчик
DallasTemperature sensors(&oneWire);
```

- Вимірювання температури:

cpp

```
sensors.begin();  // Ініціалізація датчика
sensors.requestTemperatures();  // Запит на вимірювання
float temperature = sensors.getTempCByIndex(0);  // Отримання температури
```

3. Відображення даних:
    - Виміряна температура може бути відображена на LCD-дисплеї, на комп'ютері через серійний порт або використана для управління процесами (наприклад, для активації системи охолодження).

Висновок

Датчик температури DS18B20 є універсальним і надійним рішенням для вимірювання температури в різних проектах. Його цифровий вихід, широкий діапазон вимірювань і простота підключення роблять його популярним вибором для багатьох розробників. У проекті програмованого зарядного пристрою для Arduino DS18B20 може використовуватися для моніторингу температури акумулятора, що допомагає запобігти перегріву та підвищує безпеку роботи пристрою. Датчик напруги та току INA226.Датчик INA226 — це високоточний цифровий сенсор, призначений для вимірювання напруги, струму та потужності в електри-

чних колах. Він розроблений для використання в системах моніторингу живлення, енергетичних приладах, а також у проектах, де важливо відстежувати енергоспоживання.

Основні характеристики INA226(Рис.2.11)

Цифровий вихід:

INA226 використовує протокол I2C для передачі даних, що дозволяє легко інтегрувати його з мікроконтролерами, такими як Arduino, Raspberry Pi та інші.

Діапазон вимірювань:

Датчик може вимірювати напругу в діапазоні до 36 Вольт, що робить його придатним для різних застосувань, від низьковольтних до високовольтних систем.

Вимірювання струму:

INA226 використовує вбудований резистор шунта для вимірювання струму, дозволяючи визначати струм до 3,2 А з точністю ±1.5%. Користувач може підключити резистор шунта з будь-яким значенням, залежно від потреб проекту.

Вимірювання потужності:

Датчик може розраховувати споживану потужність, спостерігаючи за напругою та струмом одночасно, що дозволяє отримати точні дані про енергоспоживання.

Вбудовані функції:

INA226 має вбудовані функції, такі як автоматичне вимкнення, конфігурація порогів, і інтервали вимірювання, що дозволяє знижувати споживання енергії та підвищувати точність

Основні команди:

- Ініціалізація бібліотек:

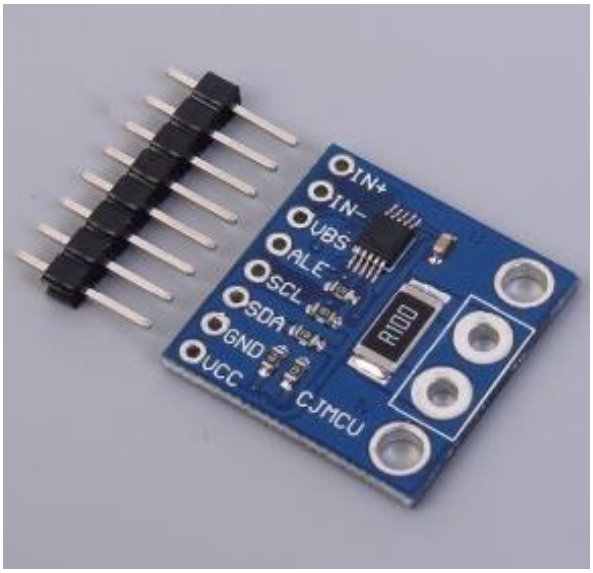


Рис.2.11 Датчик напруги та току INA226

cpp

#include <Wire.h>

#include <Adafruit_INA226.h>

Adafruit_INA226 ina226; // Створення об'єкта INA226

- Ініціалізація датчика:
- cpp

ina226.begin();

- Вимірювання напруги, струму та потужності:

cpp

float voltage = ina226.readBusVoltage(); // Читання напруги

float current = ina226.readCurrent(); // Читання струму

float power = ina226.readPower(); // Читання потужності[15]

3. Відображення даних:
    - Виміряні значення напруги, струму та потужності можна відображати на LCD-дисплеї або використовувати для контролю інших елементів системи, таких як зарядні пристрої (*Рис.2*.11).

Висновок

Датчик INA226 є потужним інструментом для моніторингу напруги, струму та потужності в електричних колах. Його цифровий вихід, висока точність та простота використання роблять його ідеальним вибором для проектів, пов'язаних із енергетичним моніторингом. У проекті програмованого зарядного пристрою для Arduino INA226 може використовуватися для контролю та вимірювання споживаної енергії, що допомагає оптимізувати процес зарядки акумуляторів і підвищує ефективність пристрою[9].

2.3 Використання KiCad EDA для розробки схеми та макету друкованої плати

KiCad EDA — це потужний інструмент для розробки електронних схем та друкованих плат (PCB). Він є безкоштовним і відкритим програмним забезпеченням, що широко використовується професіоналами та аматорами для створення проектів різного рівня складності. У рамках даного проекту платформа KiCad була використана для створення електричної схеми зарядного пристрою та його фізичного макету друкованої плати[10].

Етапи створення схеми та макета плати

1. Розробка електричної схеми

   Етап розробки починається у модулі Eeschema, де було створено електричну схему пристрою:

   - Додавання компонентів: З бібліотек KiCad були вибрані основні елементи схеми, такі як мікроконтролер ATmega328P, MOSFET транзистори, резистори, конденсатори, роз'єми живлення та датчики струму й напруги.
   - З’єднання компонентів: Використовуючи інструмент трасування з’єднань, було сформовано логічні зв’язки між елементами схеми. Наприклад, підключення виводів мікроконтролера до периферійних компонентів (дисплей, сенсори тощо).

- Додавання додаткових модулів: Було включено зовнішній стабілізатор живлення для забезпечення стабільної роботи плати.
- Перевірка схеми: Вбудований модуль перевірки електричних правил (ERC) дозволив уникнути помилок, таких як відсутність з'єднань чи дублювання.

2. Розробка макету друкованої плати
   Для фізичної реалізації схеми використовується модуль Pcbnew, де було розроблено макет плати:
   - Імпорт компонентів: Всі елементи та з'єднання зі схеми автоматично імпортуються в макет плати.
   - Розташування компонентів: Компоненти були розміщені на платі з урахуванням оптимального маршруту з'єднань і мінімізації перехресних трас. Наприклад, мікроконтролер був розміщений у центрі, оточений периферійними елементами.
   - Трасування доріжок: Використовуючи напівавтоматичний інструмент трасування, було створено електричні доріжки між компонентами. Для зменшення перешкод враховувалась ширина доріжок і відстань між ними.
   - Додавання силових та заземлювальних площин: Було реалізовано площини GND та VCC для зменшення опору доріжок і поліпшення електромагнітної сумісності.
   - Перевірка макету: За допомогою перевірки правил дизайну (DRC) було усунуто можливі помилки, наприклад, перетини доріжок або недостатні відстані між компонентами[23].
3. Генерація файлів для виробництва
   Завершивши розробку, було створено файли Gerber, необхідні для виготовлення друкованої плати. Крім того, створені файли для автоматичного монтажу компонентів (Pick and Place)(*Рис. 2.12*.Рис.2.*13*)

Переваги використання KiCad

1. Інтуїтивний інтерфейс
   KiCad EDA має зручний і зрозумілий інтерфейс, що спрощує роботу навіть для новачків. Інструменти для роботи зі схемами та платами легко доступні та добре структуровані.
2. Гнучкі бібліотеки компонентів
   Програмне забезпечення включає велику базу бібліотек компонентів, яка регулярно оновлюється. Крім того, є можливість створювати власні символи та посадкові місця.
3. Моделювання та 3D-візуалізація
   Модуль 3D-візуалізації дозволив оцінити кінцевий вигляд плати, розташування компонентів і можливі конфлікти в дизайні.
4. Безкоштовність та відкритість
   Відсутність ліцензійних обмежень та доступність вихідного коду роблять KiCad чудовим вибором для розробників будь-якого рівня.

Висновок

Використання KiCad EDA для створення схеми та макета друкованої плати зарядного пристрою забезпечило зручність у проектуванні, точність і надійність. Інструменти перевірки помилок та генерації виробничих файлів дозволили мінімізувати ризик проблем під час виготовлення плати, що особливо важливо для функціонально складних пристроїв, таких як програмовані зарядні системи.(Додаток А.)

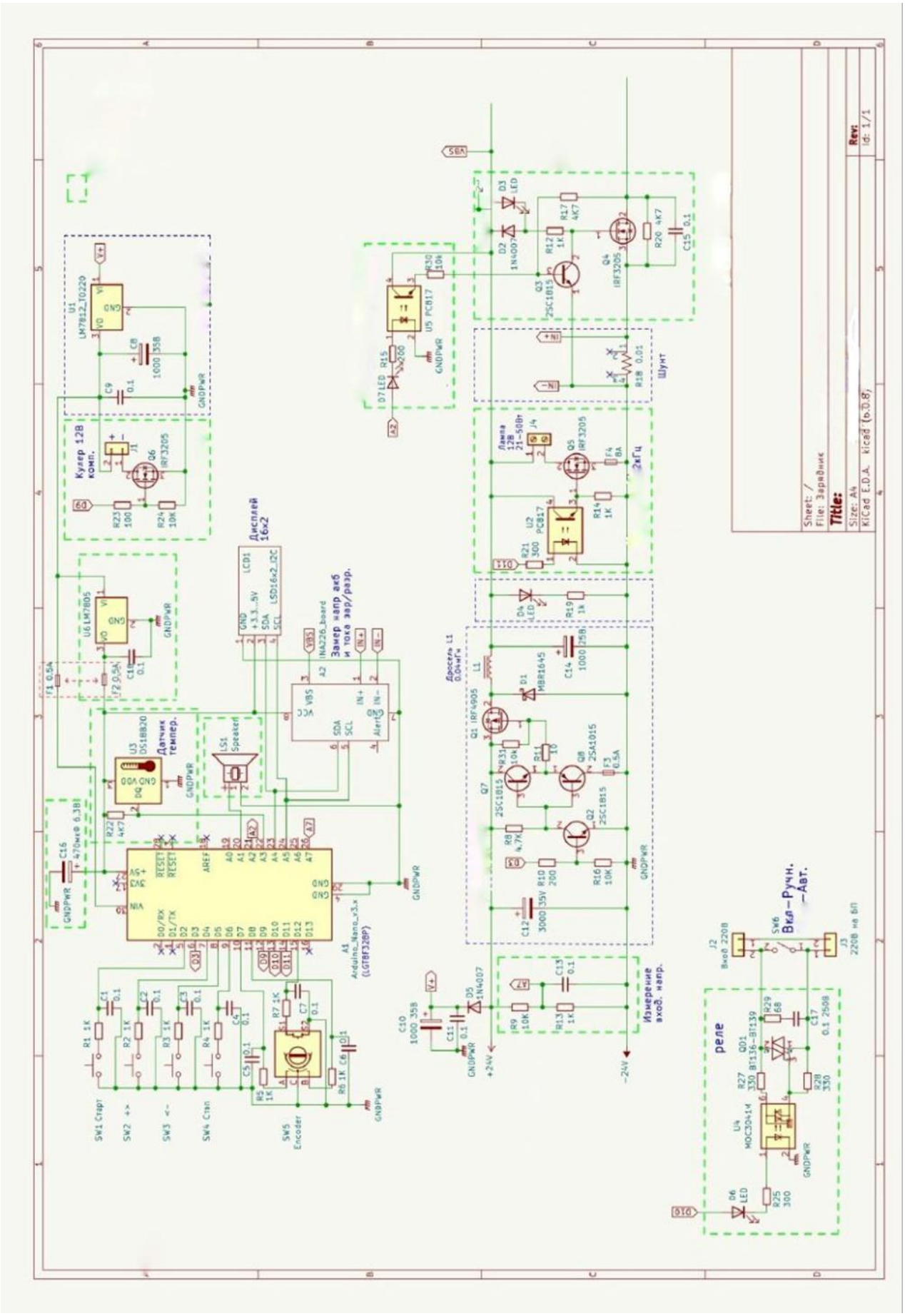


Рис. 2.12 Схема зарядного пристрою

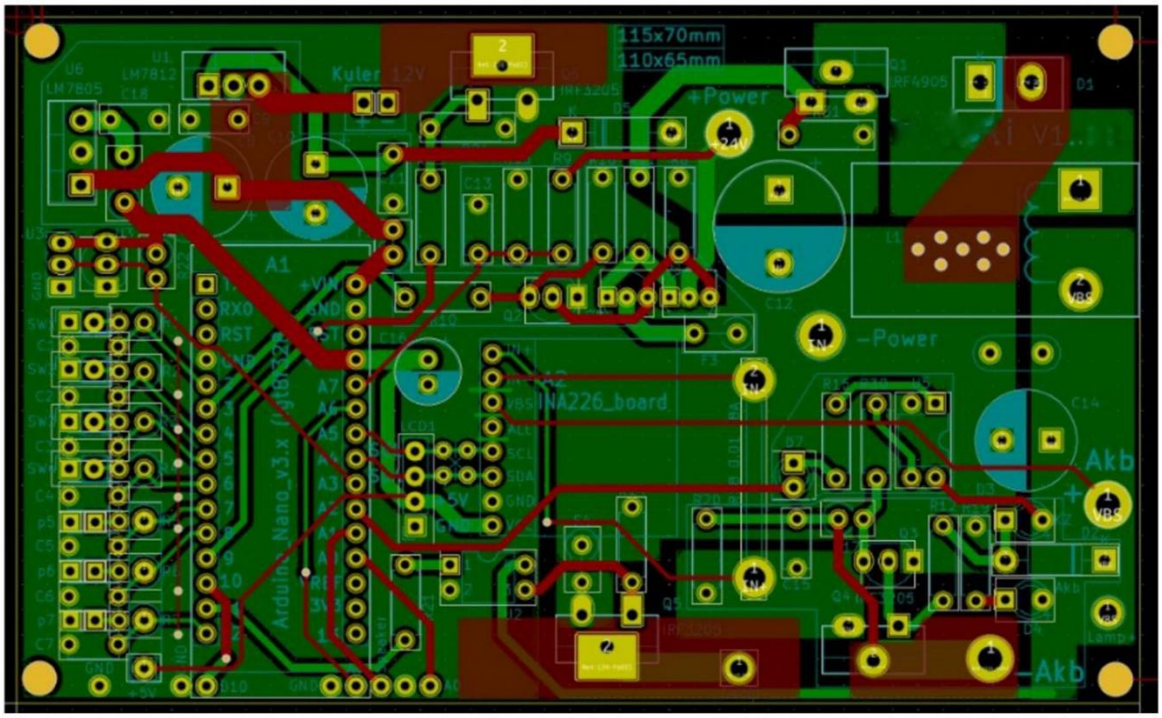


Рис.2.13 Макет плати для друку

2.4 Розробка програмного забезпечення для мікроконтролера:

Програмне забезпечення (ПЗ) мікроконтролера є важливим елементом, який забезпечує інтеграцію всіх апаратних компонентів в єдину систему. Основні функції ПЗ включають:

1. Збір і обробка даних від датчиків: Мікроконтролер постійно отримує дані від датчиків струму та напруги, аналізує ці значення і приймає рішення щодо регулювання струму заряду. Для цього застосовуються алгоритми цифрової обробки сигналів, які дозволяють відфільтровувати шум та отримувати точні дані.
2. Реалізація алгоритмів заряду: Програмне забезпечення містить кілька алгоритмів заряду для різних типів АКБ, включаючи:
    - Попередній заряд сильно розряджених батарей, який застосовується для поступового відновлення їхньої ємності.
    - Основний заряд з комбінованим методом, де спочатку стабілізується струм, а на кінцевому етапі стабілізується напруга.

- Режим "качелі", що дозволяє підтримувати батарею у зарядженому стані, зменшуючи ризики перенапруги.
- Підзарядка та режим зберігання, що підтримує батарею у належному стані під час тривалого простою.

3. Контроль температури та безпеки: Для безпеки заряду ПЗ використовує дані від датчиків температури та інших сенсорів для запобігання перегріву, який може викликати пошкодження акумулятора. Якщо температура перевищує критичне значення, мікроконтролер припиняє процес заряду або перемикається на безпечний режим.
4. Взаємодія з користувачем: ПЗ також відповідає за відображення інформації на дисплеї та прийняття команд від користувача через енкодер і кнопки. Важливо, щоб інтерфейс був інтуїтивно зрозумілим та зручним у використанні.
5. Пам'ять і збереження даних: Під час заряду акумулятора мікроконтролер збирає та зберігає дані про об'єм заряду, кількість ампер-годин, спожиті ват-години та інші параметри. Ці дані можуть бути використані для аналізу стану батареї та планування її обслуговування.
6. Автоматизація процесу: Основна мета програмного забезпечення – автоматизація процесу заряду, мінімізація необхідності втручання користувача та оптимізація умов для тривалої експлуатації АКБ. Програмне забезпечення здатне самостійно визначати тип акумулятора та застосовувати відповідний алгоритм заряду.

Таким чином, розробка ПЗ для мікроконтролера є критично важливою частиною системи управління зарядним процесом, забезпечуючи високий рівень ефективності та безпеки для акумуляторних батарей різних типів.

Програмування Arduino

Для програмування платформи Arduino використовується спрощена версія мови програмування C++, яка надає обумовлені функції для роботи з апаратним забезпеченням. Основною архітектурною одиницею програм Arduino є скетч, який містить принаймні дві ключові функції: setup() та loop()[1].

Функція setup() викликається на початку виконання скетчу та слугує для ініціалізації змінних, визначення режимів роботи виходів, встановлення з'єднання з додатковими модулями та налаштування підключених бібліотек. Ця функція запускається лише один раз після подачі живлення або скидання плати Arduino. Це дозволяє налаштувати пристрій на початкових етапах роботи.

Функція loop() виконується безпосередньо після функції setup(). Як випливає з назви, весь код, розташований між фігурними дужками в loop(), виконується в циклі, що дозволяє програмі виконувати обчислення та реагувати на них. Наприклад, мікроконтролер ATmega328, що є основним у більшості плат Arduino, може виконувати функцію loop приблизно 10,000 разів на секунду (якщо не використовуються затримки або складні обчислення). Це забезпечує високу швидкість обробки даних і можливість реагування на зміни в реальному часі.

Основними типами функцій, доступними в середовищі Arduino, є функції цифрового, аналогового та розширеного введення/виведення, а також функції для роботи з часом[11,12,13].

До цифрових функцій відносять:

- pinMode() — ця функція налаштовує режим роботи вказаного виводу, встановлюючи його вхідним (INPUT) або вихідним (OUTPUT). Це дозволяє контролювати, як вивід буде взаємодіяти з підключеними компонентами.
- digitalWrite() — ця функція дозволяє відправляти на цифровий вивід значення HIGH (високий рівень) або LOW (низький рівень). Якщо вивід налаштований як вихід (OUTPUT), то при відправленні HIGH напруга на виході зміниться на 5 В (або 3.3 В для плат, що працюють від 3.3 В), а при відправленні LOW — на 0 В. Якщо вивід налаштований як вхід (INPUT), то відправлення HIGH підключить внутрішній підтягувач резистором номіналом 20 кОм, а LOW відключить його.
- digitalRead() — функція, яка зчитує рівень сигналу HIGH або LOW з зазначеного цифрового виводу, дозволяючи контролювати стан вхідних сигналів.

Аналогові функції включають:

- analogWrite() — ця функція формує задану напругу на виході у вигляді сигналу широтно-імпульсної модуляції (ШІМ). Після виклику analogWrite, на виході буде безперервно генеруватися ШІМ-сигнал із заданим коефіцієнтом заповнення до наступного виклику цієї функції. Частота ШІМ сигналу становить приблизно 490 Гц, що дозволяє регулювати середнє значення напруги на виході.
- analogRead() — ця функція зчитує величину напруги з зазначеного аналогового виводу, що дозволяє отримувати інформацію про змінні параметри, такі як температура або освітленість. За допомогою функції analogReference() можна змінювати вхідний діапазон і роздільну здатність зчитування.
- analogReference() — функція, яка встановлює джерело опорної напруги, що використовується при зчитуванні аналогового сигналу. Це визначає максимальне значення вхідного діапазону, що є критично важливим для точної роботи з аналоговими сигналами.

До розширеного введення/виведення відносять:

- tone() — ця функція генерує на виводі прямокутний сигнал заданої частоти. Вона також дозволяє вказати тривалість сигналу. Якщо тривалість не вказана, сигнал буде генеруватися доти, поки не буде викликана функція noTone(). Слід зазначити, що одночасно може генеруватися тільки один сигнал заданої частоти.
- noTone() — функція, яка припиняє генерування прямокутного сигналу після виклику tone(). Якщо сигнал не генерується, виклик цієї функції не призводить до жодного ефекту.

До функцій роботи з часом належать:

- millis() — функція, що повертає кількість мілісекунд, які пройшли з моменту старту програми на Arduino. Ця інформація може використовуватися

для створення таймерів або управління затримками в програмі. Варто зазначити, що значення, яке повертається, переповниться (скинеться в 0) приблизно через 50 днів безперервної роботи.

- delay() — функція, яка призупиняє виконання програми на вказаний проміжок часу (в мілісекундах). Ця функція корисна для створення пауз між командами або затримок у виконанні програми.

Загалом, програмування Arduino забезпечує широкий спектр можливостей для створення різноманітних електронних проектів, що взаємодіють із фізичним середовищем, завдяки простоті використання мови програмування та потужним функціям, доступним у бібліотеках Arduino[4].

Програмування Arduino Atmega 328

Програмування мікроконтролера Arduino Atmega 328 здійснюється за допомогою середовища розробки Arduino IDE, яке підтримує просту й інтуїтивно зрозумілу мову програмування, засновану на C/C++. Для створення програми керування зарядним процесом необхідно врахувати кілька основних функцій[16]:

1. Читання датчиків: регулярний збір даних про напругу, струм і температуру батареї.
2. Управління зарядом: контроль струму і напруги на різних етапах заряду для забезпечення стабільного і безпечного процесу.
3. Аварійні режими: автоматичне вимкнення при перевищенні граничних значень температури або напруги.
4. Виведення інформації: відображення поточних параметрів на дисплеї для надання користувачу інформації про стан АКБ.
5. Збереження даних: запис параметрів заряду для їх подальшого аналізу та оптимізації процесу.

Висновок

Мікроконтролер Arduino Atmega 328 є ідеальним вибором для розробки зарядного пристрою для АКБ. Завдяки своїм характеристикам та можливостям програмування, цей контролер дозволяє реалізувати всі необхідні функції системи

управління зарядом акумуляторних батарей, забезпечуючи ефективність, безпеку та автоматизацію процесу[5].

2.4 Методи і засоби програмної підтримки вбудованої комп'ютеризованої системи для універсального зарядного пристрою

Програмна підтримка вбудованої комп'ютеризованої системи є ключовим етапом у створенні універсального зарядного пристрою, оскільки вона забезпечує взаємодію між апаратною частиною пристрою та користувачем. Методи і засоби програмної підтримки включають розробку коду, який реалізує функціональність пристрою, його налаштування та можливість адаптації до змін у технічних характеристиках(Додаток Ж).

Основні етапи розробки програмної підтримки

1. Аналіз функціональних вимог
   Перед написанням коду визначаються ключові функції зарядного пристрою, такі як:
   - Управління струмом та напругою зарядки.
   - Контроль температури компонентів і акумулятора.
   - Управління охолоджувальним вентилятором.
   - Підтримка інтерфейсу користувача для налаштувань.
2. Розробка алгоритмів
   На основі функціональних вимог створюються алгоритми управління, наприклад:
   - Алгоритм зарядки акумулятора за методом Браніміра.
   - Автоматичний розрахунок струму залежно від опору шунта.
   - Інтеграція захисту від перенапруги, перегріву чи перевищення допустимого струму.

3. Вибір програмних засобів
   Для написання коду використовуються засоби розробки, які відповідають специфіці мікроконтролера:
   - Мова програмування: Переважно використовується мова C/C++ через її оптимальність для вбудованих систем.
   - Інтегроване середовище розробки (IDE): Наприклад, Arduino IDE або PlatformIO.
4. Розробка програмного коду
   Ключовий процес програмної підтримки — написання коду. Він включає:
   - Налаштування параметрів пристрою через макроси (#define).
   - Реалізацію функцій для обробки сигналів від датчиків, наприклад, температури чи напруги.
   - Розробку програмних модулів для керування реле, вентиляторами та іншими периферійними пристроями.
5. Тестування та налагодження
   Написаний код перевіряється на тестовому обладнанні для виявлення можливих помилок. Це включає: симуляцію роботи пристрою перевірку коректності взаємодії між апаратними компонентами та програмними модулями, оптимізацію коду для зменшення затримок і ресурсоємності.

Методи програмної підтримки

- Модульний підхід
  Код розбивається на окремі модулі (керування зарядкою, вимірювання температури, управління вентилятором тощо), що спрощує його розробку, налагодження та оновлення.

- Використання переривань
  Для обробки критично важливих подій (наприклад, перевищення допустимого струму або температури) застосовуються апаратні та програмні переривання.
- Використання таймерів і ШІМ (широтно-імпульсної модуляції)
  Таймери забезпечують точність керування процесами зарядки, а ШІМ дозволяє регулювати швидкість вентилятора.
- Калібрування
  Реалізовані програмні механізми дозволяють коригувати напругу та струм відповідно до особливостей конкретного обладнання.

Засоби реалізації

- Програмні бібліотеки
  Для прискорення розробки використовуються бібліотеки для роботи з дисплеями, датчиками температури, енкодерами та іншими модулями (наприклад, LiquidCrystal_I2C для дисплеїв).
- Інтеграція EEPROM
  Для збереження налаштувань користувача використовується пам'ять EEPROM, що дозволяє зберігати параметри між циклами увімкнення/вимкнення.

Інтерфейс користувача. Включення простого меню, де користувач може налаштувати параметри пристрою через кнопки чи енкодер.

Таким чином, методи і засоби програмної підтримки вбудованої системи універсального зарядного пристрою забезпечують ефективну інтеграцію програмного забезпечення з апаратними компонентами, адаптивність пристрою до різних умов експлуатації та зручність для кінцевого користувача(Додаток Ж).

Визначення версії прошивки та контролера
Вказав версію прошивки, яка зберігається в EEPROM. Для мікроконтролерів "Ver 1.02-328p".

```
// 1. Визначення версії прошивки та контролера
#define MEMVER 102 // версія прошивки в EEPROM
#if defined(__LGT8FX8P__)
  #define VER "Ver 1.02-lgt8" // версія прошивки для LGT8FX8P
#else
  #define VER "Ver 1.02-328p" // версія прошивки для інших контролерів
#endif
```

1. Вибір мови інтерфейсу
   Налаштував мову: 1 — англійська, 2 — трансліт.

   ```
   #define INTERFACE 1 // 1 - англійська, 2 - трансліт
   ```

2. Налаштування параметрів дисплея
   - Задав адресу дисплея (0x27 або 0x3F).
   - Вказав кількість стовпців (16 чи 20) та рядків (2 чи 4).

     ```
     #define ADDRDISP 0x27 // адреса дисплея
     #define DISPLAYx 16   // кількість стовпців
     #define DISPLAYy 2    // кількість рядків
     ```

3. Вибір типу управління
   Задав спосіб управління: тільки енкодер, кнопки, або комбінація. Використав значення 2 для енкодера з кнопками Пуск/Стоп.

   ```
   #define ENCBUTT 2 // 0 - тільки енкодер, 1 - тільки кнопки, 2 - енкодер + Пуск + Stop
   ```

4. Тип спікера
   Обрав пасивний спікер (1) або активний (2).

   ```
   #define SPEAKER 1 // 1 - пасивний, 2 - активний
   ```

5. Налаштування функціоналу

   ```
   #define VOLTIN 1   // замір напруги БП
   #define FAN 1      // управління вентилятором
   #define PROT 1     // управління модулем захисту
   #define DISCHAR 1  // розрядний модуль
   ```

#define RESIST 1 // вимірювання опору

#define BRANIMIR 1 // заряд за методикою Браніміра

6. Реле або індикатор БП
   Визначив тип реле (220В) або індикатора (для БП).

   #define POWPIN 0 // 0 - не використовується, 1 - реле 220В, 2 - індикатор БП

7. Опір шунта
   Встановив значення опору шунта (0.01085 Ом) для розрахунку максимального струму зарядного пристрою.

   #define SHUNT 0.01085 // опір шунта, Ом

   #define CURR_MAX (0.08192f / (float)SHUNT) // Максимальний струм

   //#define CURR_MAX 8.0 // Встанови вручну при потребі

8. Корекція напруги
   Увімкнув корекцію з урахуванням опору лінії до акумулятора (61 мОм).

   #define OHMS 61 // опір лінії до АКБ, мОм

9. Калібрування напруги
   Задав опорну напругу (5000 мВ) і параметри резистивного подільника (верхній резистор — 10000 Ом, нижній — 1000 Ом).

   #define VREF 5000 // опорна напруга, мВ

   #define DIV_R1 10000 // верхній резистор, Ом

   #define DIV_R2 1000 // нижній резистор, Ом

10. Датчики температури
    - Налаштував датчик температури транзистора (DS18B20).
    - Визначив параметри термістора: B-коефіцієнт (3435), опір при 25°C (10000 Ом) та підтягуючий резистор.

      #define SENSTEMP1 1 // 1 - DS18B20, 2 - NTC, 0 - не використовується

      #define NTCB1 3435 // B-коефіцієнт термістора

      #define NTCR1 10000 // опір термістора при 25°C, Ом

      #define NTCRS1 10000 // підтягуючий резистор, Ом

#define SENSTEMP2 0 // 0 - не використовується, 1 - DS18B20, 2 - NTC

#define NTCB2 3435 // B-коефіцієнт термістора

#define NTCR2 10000 // опір термістора при 25°C, Ом

#define NTCRS2 10000 // підтягуючий резистор, Ом

11. Управління вентилятором
Встановив температурні пороги для включення/вимкнення вентилятора та обмеження струму:
    - Включення — при 40°C.
    - Вимкнення — при 36°C.
    - Максимальні оберти — при 60°C.

    #define DEG_ON 40 // температура включення вентилятора, °C #define DEG_OFF 36 // температура вимкнення вентилятора, °C #define DEG_MAX 60 // температура максимальних обертів, °C #define DEG_CUR 70 // температура обмеження струму, °C #define CURFAN_ON 1500 // струм для включення вентилятора, мА #define CURFAN_OFF 1200 // струм для вимкнення вентилятора, мА #define CURFAN_MAX 7000 // струм для максимальних обертів вентилятора, мА

12. Струм вентилятора

    Задав пороги струму для управління обертами вентилятора:
    - Включення — 1500 мА.
    - Вимкнення — 1200 мА.
    - Максимальні оберти — 7000 мА.

Параметри управління вентилятором

#define DEG_ON 40 // температура включення вентилятора у градусах

#define DEG_OFF 36 // температура відключення вентилятора у градусах

#define DEG_MAX 60 // температура максимальних обертів вентилятора

#define DEG_CUR 70 // температура при якій зменшується струм заряду

#define CURFAN_ON 1500 // значення струму в міліамперах при якому включається вентилятор. (До 32А)

#define CURFAN_OFF 1200 // значення струму в міліамперах при якому відключається вентилятор. (До 32А)

#define CURFAN_MAX 7000 // значення струму в міліамперах для максимальних обертів вентилятора. (До 32А)

14. Значення напруги БП
    Значення напруги БП

#define POWER 24 // 24 - напруга від блока живлення, вольт(ціле число). Якщо встановлено дільник напруги на вході, то заміряється в програмі. (Цифрове обмеження 65 Вольт)

#define POWER_MAX 26 // максимальна допустима напруга від БП (17-32 Вольт). Якщо воно перевищить, то відключиться реле 220В (пін 10) на 10 секунд (якщо був просто стрибок напруги). Утитуй напругу C12 та C10 та стабілізатора LM7812.

У процесі програмування меню для універсального зарядного пристрою було реалізовано функцію Menu1602(), яка відповідає за взаємодію з користувачем через LCD-дисплей та забезпечує доступ до основних налаштувань пристрою. Нижче наведено ключові аспекти написання цієї функції, її структуру та особливості роботи.

Опис виконання

1. Рівні меню Функція підтримує кілька рівнів меню (urovenmenu). Основне меню дозволяє перемикатися між пунктами, а підменю надає доступ до параметрів, які можна змінювати.
2. Циклічна навігація Перемикання між пунктами реалізовано за допомогою кругового механізму, що спрощує використання інтерфейсу.
3. Оновлення дисплея При зміні пункту меню або параметра дисплей оновлюється через функцію lcd.clear() та виклик відповідних методів для виводу даних (print_Capacity, print_mode, Vout).
4. Перевірка стану Перед запуском налаштувань перевіряється стан зарядного пристрою за допомогою прапора CHARGE.
5. Розрахунки У підменю проводяться розрахунки струму та напруги для зарядки, ро рядки, дозарядки тощо. Наприклад, струм заряду обмежується через функцію constrain, щоб запобігти перевищенню допустимих значень.
6. Обробка помилок Якщо вибрані некоректні параметри (наприклад, невірний тип акумулятора), функція відображає повідомлення про помилку на дисплеї.

Функція розряду акумулятора (Discharge())

Реалізує контрольований розряд АКБ:

1. Відображення параметрів: виводить напругу, струм та кількість циклів на дисплей.

cpp

PrintVA(pam.Volt_discharge, pam.Current_discharge, 0, 0, 1);

printCykl();

2. Розряд: запускає модуль розряду з заданою частотою (4 кГц).

cpp

Freq(vkr.service[FREQDISCHAR]);

3. Моніторинг стану: контролює час, струм і напругу для зупинки розряду при досягненні меж.

cpp

if (ina.voltsec <= pam.Volt_discharge and ampsec <= pam.Current_discharge_min) dischar = false;

4. Обробка подій: обробляє натискання кнопок, керує дисплеєм і вентилятором.

cpp

switch (tiks) { case ENCHELD: bitClear(pam.MyFlag, CHARGE); break; }

5. Розрахунок енергії: обчислює ампер-години та ват-години протягом розряду.

cpp

pam.Ah_discharge += aw_ch(abs(ina.ampersec));

pam.Wh_discharge += aw_ch(abs(ina.getPower()));

Функція вимірювання опору (Resist())

Вимірює внутрішній опір АКБ:

1. Початковий контроль: перевіряє напругу та встановлює струм розряду.

cpp

Fixcurrent(400);

2. Два виміри: записує напругу й струм перед увімкненням максимального струму і після нього.

cpp

I[0] = abs(ina.ampersec); U[0] = ina.voltsec;

digitalWrite_speed(PWMDCH, HIGH);

3. Розрахунок опору: обчислює опір за формулою R=ΔUΔIR = \frac{\Delta U}{\Delta I}R=ΔIΔU.

cpp

uint16_t r = (U[0] - U[1]) * 100000 / (I[1] - I[0]);

4. Результати: виводить значення опору та пусковий струм на дисплей.

cpp

pam.Resist_2 = (uint16_t)((uint32_t)bat.Volt(3) * 100 / r);

Обидві функції забезпечують точний контроль параметрів АКБ, підтримують безпечну експлуатацію та зручне відображення даних на дисплеї.

Основна структура класу INA226:

Клас надає інструменти для роботи з модулем INA226 через I2C. Він включає методи ініціалізації, налаштування, зчитування даних і обробки сигналів тривоги.

Конструктор:

cpp

INA226(uint8_t address)

: _iic_address(address) {}

- Приймає: I2C-адресу модуля (наприклад, 0x41).
- Призначення: Ініціалізує об’єкт із заданою адресою.

Метод begin():

cpp

bool begin() {

Wire.begin(); // Запускає I2C-шину

if (!testConnection()) return false; // Перевіряє підключення до модуля

```
  writeRegister(INA226_REG_MASKENABLE, INA226_BIT_SOL);
  writeRegister(INA226_REG_ALERTLIMIT, 32400);


  return true;
}
```

1. Ініціалізація I2C:
    - Wire.begin(): Готує шину до передачі даних.
2. Перевірка з'єднання:
    - Використовується testConnection(), щоб упевнитися, що модуль відповідає.
3. Налаштування сигналу тривоги:
    - INA226_REG_MASKENABLE: Активує біт сигналу SOL, який відповідає за спрацювання при перевищенні напруги на шунті.
    - INA226_REG_ALERTLIMIT: Встановлює граничне значення для сигналу тривоги. У прикладі використовується значення 32400, що відповідає напрузі 81 мВ81 \, \text{мВ}81мВ при шунті 0.01 Ω0.01 \, \Omega0.01Ω.
4. Результат:
    - Якщо всі операції виконано успішно, метод повертає true.

Зчитування даних:

- Напруга на шунті:

cpp

Копіювати код

float getShuntVoltage();

Повертає значення напруги на шунті.

- Напруга на шині:

cpp

float getVoltage();

Зчитує напругу живлення.

- Струм:

cpp

float getCurrent();

Повертає обчислений струм (за допомогою шунтового опору).

- Потужність:

cpp

float getPower();

Вираховує потужність як добуток струму та напруги.

Калібрування:

- Зчитування калібрувального значення:

cpp

uint16_t getCalibration();

Отримує значення калібрування, яке зберігається в реєстрі.

- Установка калібрування:

cpp

void setCalibration(calibration value);

Дозволяє записати значення калібрування.

- Регулювання калібрування:

cpp

void adjCalibration(calibration offset);

Надає можливість тонко налаштовувати калібрування.

Налаштування часу вибірки:

- Метод setSampleTime(ch, time):

cpp

void setSampleTime(ch, time);

Встановлює тривалість вибірки для каналів:

    - INA226_VBUS (напр. шина) або
    - INA226_VSHUNT (напр. шунт).

Час вибірки вибирається з набору, наприклад, INA226_CONV_1100US (1.1 мс).

Усереднення вимірювань:

- Метод setAveraging(avg):

cpp

void setAveraging(avg);

Встановлює кількість усереднень:

  - Значення від INA226_AVG_X1 (без усереднень) до INA226_AVG_X1024 (1024 вимірювання).

Особливості сигналів тривоги:

- Реєстр INA226_REG_MASKENABLE:
  - Включає сигнали, які активують ніжку Alert.
  - Наприклад, INA226_BIT_SOL відповідає за спрацьовування при перевищенні напруги.
- Реєстр INA226_REG_ALERTLIMIT:
  - Встановлює значення межі, при перевищенні якої спрацьовує сигнал.

    Налаштування INA 226

    #include "Arduino.h"

  #include <microWire.h>

    Конструктор:

    INA226 ina226 (Адреса на шині I2c)

    INA226 ina226 (0x41); // Шунт та макс. стандартний струм, адреса 0x41 - підійде для декількох модулів

    Методи:

    bool begin(); // Ініціалізація модуля та перевірка присутності, поверне false якщо INA226 не знайдена

    void sleep(true/false); // Вмикання та вимикання режиму низького енергоспоживання, залежно від аргументу

    void setAveraging(avg); // Установка кількості усереднень вимірів (див. таблицю нижче)

void setSampleTime(ch, time); // Установка часу вибірки напруги та струму (INA226_VBUS / INA226_VSHUNT), за замовчуванням INA226_CONV_1100US

float getShuntVoltage(); // Прочитати напругу на шунті

float getVoltage(); // Прочитати напругу

float getCurrent(); // Прочитати струм

float getPower(); // Прочитати потужність

uint16_t getCalibration(); // Прочитати калібрувальне значення (після старту розраховується автоматично)

void setCalibration(calibration value); // Записати калібрувальне значення (можна зберігати його в EEPROM)

void adjCalibration(calibration offset); // Підкрутити значення калібрування на вказане значення (можна зміню-вати на ходу)

/* Public-визначення (константи) */

#define INA226_VBUS true // Канал АЦП, що вимірює напругу шини (0-36в)

#define INA226_VSHUNT false // Канал АЦП, що вимірює напругу на шунті

// Час вибірки (накопичення сигналу для оцифрування)

#define INA226_CONV_140US 0b000

#define INA226_CONV_204US 0b001

#define INA226_CONV_332US 0b010

#define INA226_CONV_588US 0b011

#define INA226_CONV_1100US 0b100

#define INA226_CONV_2116US 0b101

#define INA226_CONV_4156US 0b110

#define INA226_CONV_8244US 0b111

Створення об'єкта: INA226 ina226(0x41); — створюється об'єкт модуля з вказівкою I2C-адреси, яка підходить для підключення кількох пристроїв.

Ініціалізація модуля: Метод begin() перевіряє, чи доступний модуль. У разі відсутності з'єднання програма зупиняється.

Налаштування параметрів: Методи setAveraging() та setSampleTime() дозволяють вибрати оптимальні параметри для стабільності та точності вимірювань.

Калібрування: setCalibration(1024) встановлює значення, що відповідає конкретному опору шунта. Це необхідно для точного обчислення струму.

Зчитування даних: Методи getVoltage(), getShuntVoltage(), getCurrent() та getPower() використовуються для отримання даних про стан електричної системи. Отримані значення можуть бути передані для подальшої обробки або відображення.

Режим роботи: При потребі можна використовувати sleep(true) для переведення модуля в режим низького енергоспоживання.

Функція Discharge() реалізує контрольований процес розряду акумулятора. Вона включає такі ключові етапи:

1. Ініціалізація режиму розряду:
    - Встановлюється частота роботи розрядного модуля.
    - Увімкнення вентилятора для охолодження.
    - Запуск вимірювальних модулів (INA226) та таймерів.
2. Моніторинг процесу:
    - Постійний контроль напруги, струму та температури.
    - Регулювання струму розряду через ШІМ.
3. Виведення інформації:
    - Відображення параметрів розряду на дисплеї.
    - Запис даних про стан акумулятора, таких як ампер-години (Ah), ват-години (Wh), час розряду.
4. Завершення процесу:
    - Автоматичне вимкнення розряду при досягненні встановлених меж напруги та струму.
    - Ручне завершення через натискання кнопок.

```
#if (DISCHAR == 1)
// Функція розряду акумулятора
```

```
void Discharge() {
  // Виведення режиму, напруги та струму
  PrintVA(pam.Volt_discharge, pam.Current_discharge, 0, 0, 1);
  printCykl(); // Відображення кількості циклів
  Freq(vkr.service[FREQDISCHAR]); // Частота роботи розрядного модуля (4 кГц)
  const uint16_t s1 = ina.voltsec; // Поточна напруга
  uint8_t tok = 1, n_disp = 0, ns_disp = 0;
  int8_t tiks = 0;
  bool dischar = true; // Розряд увімкнено
uint8_t time_millis = 0;
  #if (LOGGER > 0)
  uint8_t logg = LOGGTIME; // Період передачі даних
  #endif
  Delay(3000); // Затримка перед початком
  int8_t tr_c = 0; // Температура
  #if (FAN && KULDISCHAR == 2)
  digitalWrite_speed(PWMKUL, HIGH); // Увімкнення вентилятора
  #endif
  #if (VOLTIN == 1)
  vin.start(); // Запуск вимірювання напруги
  #endif
  uint16_t time10 = 600; // 10 хвилинний інтервал
  disp.start(); // Старт таймера
  ina.start(2); // Запуск вимірювань INA226
  dcdc.start();
  // Основний цикл розряду (до 35 годин)
  while (dischar && bitRead(pam.MyFlag, CHARGE) && pam.Time_discharge < 126000) {
    tiks = myTick(); // Опитування кнопок
```

```
if (tiks) {
  // Обробка натискання кнопок
  switch (tiks) {
    case ENCHELD:
    case STOPHELD:
      bitClear(pam.MyFlag, CHARGE); // Завершити розряд
      break;
    case ENCRIGHT:
    case ENCLEFT:
      n_disp++; // Змінити відображення
      ns_disp = TIME_DISP;
      lcd.clear();
      break;
  }
  Light1(true);
}
// Регулювання напруги та струму
if (ina.sample()) {
  dcdc.Control_dich(); // Контроль струму розряду
  #if (VOLTIN == 1)
  vin.sample(); // Оновлення напруги
  #endif
}
if (disp.tick()) time_millis = 5;
// Щосекундна обробка
if (time_millis) {
  switch (time_millis) {
    case 5:
      pam.Ah_discharge += aw_ch(abs(ina.ampersec)); // Розрахунок Ah
      pam.Wh_discharge += aw_ch(abs(ina.getPower())); // Розрахунок Wh
```

```
            pam.Time_discharge++; // Час розряду

            if (ina.voltsec <= pam.Volt_discharge) dischar = false; // Завершення розряду
            break;
          case 4:
            tr_c = read_tq1(); // Читання температури
            break;
          case 3:
            // Відображення даних на дисплеї
            Display_print(pam.Ah_discharge, pam.Wh_discharge, pam.Time_discharge, tr_c, map(ina.voltsec, pam.Volt_discharge, s1, 0, 100));
            break;
        }
      }
    }}#endif(Додаток Ж)
```

Функція заряду по Браніміру реалізує специфічний метод заряджання акумуляторів, що включає контроль напруги, струму та часу, а також перемішування електроліту після завершення основного заряду. Це дозволяє забезпечити повний заряд і підтримувати акумулятор у робочому стані (Додаток Ж).

```
// Елемент коду: if (bitRead(flagsz, VMAXM))
// Застосування: Початок відліку часу, коли напруга досягає максимуму.
if (bitRead(flagsz, VMAXM)) {
  // Елемент коду: tyme_branim += 300;
  // Застосування: Додавання інтервалу часу в 5 хвилин для контролю тривалості заряду.
  tyme_branim += 300;

  // Елемент коду: if (tyme_branim >= 72000)
```

// Застосування: Перевірка, чи минуло більше 20 годин заряду. Якщо так — зупинка процесу і початок перемішування.

if (tyme_branim >= 72000) {

// Елемент коду: bitClear(flagsz, CHARGEZ);

// Застосування: Вимкнення процесу заряду після досягнення ліміту часу або завершення процесу.

bitClear(flagsz, CHARGEZ);

// Елемент коду: bitSet(pam.MyFlag, BRANIMBOIL);

// Застосування: Встановлення прапора для активації перемішування електроліту.

bitSet(pam.MyFlag, BRANIMBOIL);

}

// Елемент коду: if (va.amin[12] <= curr_min)

// Застосування: Перевірка, чи струм заряду знизився до мінімального встановленого рівня.

if (va.amin[12] <= curr_min) {

// Елемент коду: regim_end = 2;

// Застосування: Перехід до завершення заряду після виконання умов (напруга максимум, струм мінімум).

regim_end = 2;

// Елемент коду: bitSet(pam.MyFlag, BRANIM);

// Застосування: Встановлення прапора успішного завершення заряду по Браніміру.

bitSet(pam.MyFlag, BRANIM);

Розгляд етапів коду

Початок відліку часу:

При досягненні максимальної напруги зарядний пристрій починає відлік часу для контролю тривалості процесу.

Перевірка часу та завершення заряду:

Якщо заряд триває більше 20 годин, пристрій автоматично зупиняє заряд і активує функцію перемішування електроліту.

Контроль струму:

У разі, якщо струм заряду падає до мінімального рівня, процес завершується, встановлюється прапор успішного заряду.

Збереження налаштувань:

Щоб уникнути втрати даних у разі аварійного вимкнення, параметри зберігаються кожні 10 хвилин.

Втрата живлення:

Код перевіряє наявність напруги від блоку живлення та вимикає пристрій у разі її зникнення, забезпечуючи безпеку системи.

У цьому розділі було розглянуто методи і засоби розробки програмної підтримки вбудованої комп'ютеризованої системи, що є невід'ємною частиною універсального зарядного пристрою. Проаналізовано сучасні підходи до побудови програмного забезпечення для систем моніторингу та управління енергетичними процесами, обґрунтовано вибір алгоритмів для ефективного управління зарядними циклами, а також забезпечення захисту батарей від перевантажень, перегріву та перезарядження (Додаток Ж).

Для розробки програмного забезпечення зарядного пристрою було обрано модульний підхід, який спрощує процес розробки, забезпечує легкість тестування та дозволяє масштабувати систему під нові задачі. Кожен модуль відповідає за окрему функціональність, що забезпечує зручність інтеграції та налаштування.

Режим розрядки є важливою складовою тренування акумуляторів, що дозволяє оптимізувати їх ємність, оцінити стан та забезпечити тривалий термін служби. Програмне забезпечення зарядного пристрою інтегрується з інтерфейсом користувача, забезпечуючи зручний доступ до налаштувань та інформації:

Дані про напругу, струм, потужність, стан акумулятора та температуру відображаються

Також визначено основні параметри інтеграції системи в зовнішні інтерфейси для забезпечення користувацької взаємодії та віддаленого управління. Програмна частина зарядного пристрою є адаптивною та забезпечує гнучке налаштування для роботи з різними типами батарей, що значно розширює сферу застосування розробки.(Додаток В,Г).

Таким чином, розроблена програмна підтримка задовольняє вимоги до сучасних вбудованих систем управління, забезпечує високу ефективність, безпеку та зручність використання універсального зарядного пристрою.

Таким чином, розроблена програмна підтримка повною мірою відповідає сучасним вимогам до вбудованих систем управління, що використовуються у зарядних пристроях. Реалізовані алгоритми моніторингу, калібрування та управління енергетичними параметрами забезпечують високу ефективність роботи системи, мінімізуючи енергетичні втрати та підвищуючи точність вимірювань.

Інтеграція засобів контролю напруги, струму та температури з автоматизованими механізмами аналізу та захисту дозволяє досягти високого рівня надійності та безпеки під час експлуатації пристрою. Використання модульного підходу сприяє гнучкості системи, що забезпечує легкість у впровадженні нових функцій, модифікаціях та масштабуванні. Крім того, зручний інтерфейс користувача, який включає інтуїтивно зрозуміле керування, відображення інформації у реальному часі та автоматичне збереження параметрів, підвищує практичність застосування розробленого зарядного пристрою.

Розроблена система програмної підтримки універсального зарядного пристрою відповідає принципам енергоефективності, функціональної автономності та ергономічності, демонструючи потенціал для широкого впровадження в різних сферах, що вимагають точного контролю і обслуговування акумуляторних елементів.

РОЗДІЛ 4 ОПИС ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ЗАРЯДНОГО ПРИСТРОЮ

4.1 Користувацьке меню

1) "Pr:" Поточний профіль

0 - 9 — зарядний пристрій має 10 незалежних профілів. У кожному профілі зберігаються налаштування акумулятора: ємність, напруга, тип, режим роботи, А/год, Вт/год, внутрішній опір та інші параметри (напруга заряду, струм заряду тощо).

Натисни *Пуск* або *Енкодер*, обери потрібний номер профілю, повертаючи *Енкодер* або натискаючи кнопки *Плюс/Мінус*, потім знову натисни *Пуск* або *Енкодер*.

2) "Capacity Akb" Ємність акумулятора

Від 1 до 255 А·год — встанови ємність заряджуваного акумулятора. Натисни *Пуск* або *Енкодер*, встанови ємність акумулятора, повертаючи *Енкодер* або використовуючи кнопки *Плюс/Мінус*, потім знову натисни *Пуск* або *Енкодер*. При зміні ємності автоматично розраховуються:

- струм заряду;
- мінімальний струм заряду (струм завершення заряду);
- струм розряду;
- мінімальний струм розряду (струм завершення розряду);
- струм передзаряду;
- мінімальний струм передзаряду (струм завершення передзаряду);
- струм попереднього заряду.

3) "Type Akb" Тип акумулятора

Вибір типу акумулятора:

- "Pb Ca/Ca" — свинцево-кислотний акумулятор Ca/Ca з додаванням кальцію в пластини;
- "Pb Ca+" — гібридний свинцево-кислотний акумулятор Sur+ Ca/Ca з домішками кальцію;
- "Pb Sur" — свинцево-кислотний сурм'янистий акумулятор;

- "Pb AGM" — герметизований свинцево-кислотний акумулятор з абсорбованим електролітом у скляних матах;
- "Pb Gel" — герметизований свинцево-кислотний акумулятор із гелеподібним електролітом;
- "Li-ion" — літій-іонний акумулятор;
- "LiFePo4" — літій-залізо-фосфатний акумулятор;
- "LiTit" — літій-титанатний акумулятор;
- "NiCd/Mh" — нікель-кадмієвий/нікель-металогідридний акумулятор.

При зміні типу АКБ у налаштуваннях змінюється максимальна напруга заряду, дозаряду, розряду.

Режими дозаряду та заряду за Бранимиром доступні тільки для свинцево-кислотних акумуляторів.

4) "Voltage Akb" Напруга акумулятора

Встанови напругу акумулятора. Напруга задається вибором кількості осередків акумулятора. Напруга розраховується як добуток напруги одного осередку на їх кількість.

Наприклад, для Pb Ca/Ca: напруга одного осередку 2,1 В * 6 осередків = 12,6 В.

При виборі напруги акумулятора розраховуються:

- напруга заряду;
- напруга дозаряду;
- напруга буферного режиму;
- напруга зберігання;
- напруга розряду;
- напруга передзаряду.

5) "Operating mode" Режим роботи зарядного пристрою

Натисни *Пуск* або *Енкодер*, щоб вибрати режим роботи зарядного пристрою, використовуючи кнопки *Плюс/Мінус* або повертаючи *Енкодер*. Для виходу в основне меню натисни *Пуск* або *Енкодер*.

Режими роботи:

1. "Charge" Заряд — заряд постійним струмом до напруги заряду з подальшим обмеженням напруги та зменшенням струму до мінімального;
2. "Precharge" Передзаряд — для сильно розряджених акумуляторів;
3. "Charge>AddChar" Заряд > Дозаряд — спочатку заряд, потім дозаряд;
4. "Charge Branim" Заряд за Бранимиром — малими струмами;
5. "Add charge" Дозаряд — стабілізація струму до напруги дозаряду;
6. "Discharge" Розряд — розряд акумулятора;
7. "Contr.trai.cycle" Контрольно-тренувальний цикл (КТЦ) — цикл: заряд → відстій → розряд → заряд → дозаряд;
8. "Storage" Зберігання — режим зберігання після завершення заряду, дозаряду, КТЦ.

6) "Settings" Налаштування

Задайте параметри:

- "Volt charge" — напруга заряду 1,0 - БП Вольт;
- "Curr. Charge" — струм заряду 0,1 - 8,0 А;
- "Curr. charge min" — мінімальний струм заряду 0,1 - 8,0 А;
- "Time charge" — час заряду 1 - 255 год;
- "Precharge" — передзаряд: вкл/викл;
- "Volt pre charge" — напруга передзаряду 1,0 - БП Вольт;
- "Curr. pre charge" — струм передзаряду 0,1 - 8,0 А;
- "Volt add charge" — напруга дозаряду 1,0 - БП Вольт;
- "Curr. add charge" — струм дозаряду 0,1 - 8,0 А;
- "Curr addChar min" — мінімальний струм дозаряду 0,1 - 8,0 А;
- "Time add charge" — час дозаряду 1 - 255 год;
- "Volt dischar" — напруга розряду 1,0 - БП Вольт;
- "Curr. dischar" — струм розряду 0,1 - 8,0 А;
- "Curr.disch. Off" — струм завершення розряду 0,1 - 8,0 А;
- "Storage regim" — напруга режиму зберігання;
- "Bufernii regim" — напруга буферного режиму 1,0 - БП Вольт;
- "Oscillation" — режим "качелі": вкл/викл;

- "Cycle" — кількість циклів 1 - 255;
- "!system param!" — системні параметри зарядного пристрою:
    - "POWER" — напруга БП Вольт;
    - "POWER_MAX" — максимальна напруга БП Вольт;
    - "SHUNT" — опір шунта: 100 (10,0 мОм), 90 (9,0 мОм), 55 (5,5 мОм);
    - "CURR_MAX" — максимальний струм зарядного пристрою;
    - "FAN" — режим роботи вентилятора:
        - 0 — вимкнено;
        - 1 — ШІМ (плавне регулювання);
        - 2 — вкл/викл;
    - "DEG_ON" — температура увімкнення вентилятора (градуси);
    - "DEG_OFF" — температура вимкнення вентилятора (градуси);
    - "DEG_MAX" — температура для максимальних обертів вентилятора;
    - "DEG_CUR" — температура, при якій зменшується струм заряду;
    - "CURFAN_ON" — струм увімкнення вентилятора в А/10;
    - "CURFAN_OFF" — струм вимкнення вентилятора в А/10;
    - "CURFAN_MAX" — струм для максимальних обертів вентилятора (ШІМ) в А/10;
    - "TIME_LIGHT" — час підсвітки дисплея (хвилини);
    - "FREQ_CHARGE" — частота силового модуля 0-7 {0.25, 0.5, 1, 2, 4, 8, 30, 60} кГц;
    - "FREQ_DISCHAR" — частота розрядного модуля 0-7 {0.25, 0.5, 1, 2, 4, 8, 30, 60} кГц.

Скидання налаштувань ("Reset settings"):

- Exit — вийти зі скидання, не змінюючи налаштування;
- All — повне скидання всіх налаштувань і параметрів;
- Nastroiki — скидання налаштувань усіх профілів;
- Calibration — скидання калібрувань;
- !system param! — скидання системних параметрів зарядного пристрою.

7) "Resistance Akb" Вимірювання внутрішнього опору

При натисканні *Пуск* або *Енкодер* здійснюється вимірювання внутрішнього опору акумулятора. Протягом 10 секунд низьким струмом розряду (0,3–0,5 А) знімається поверхнева напруга. Потім струм розряду збільшується до максимального, і через 1 секунду проводиться вимірювання. Далі розраховується внутрішній опір акумулятора та приблизний пусковий струм.

8) "Statistics" Статистика

В цьому розділі можна переглянути дані про попередні цикли заряду, розряду, вимірювання внутрішнього опору та КТЦ.

9) "Calibration" Калібрування

Виконується калібрування основних вимірюваних параметрів.

10) Індикація

Тут відображаються:

- поточна напруга від блоку живлення;
- напруга акумулятора;
- струм від акумулятора до зарядного пристрою;
- температура всередині корпусу.(Додаток Б,В).

4.2 Режими роботи зарядного пристрою (ЗП)

Заряд ("Charge")

Заряд акумулятора проводиться постійним струмом до досягнення напруги заряду, після чого напруга обмежується, а струм зменшується до мінімального. Після закінчення встановленого часу заряд припиняється. Якщо струм знижується нижче мінімального, заряд припиняється через 10 хвилин.

Основні параметри контролюються наступним чином:

- Напруга і струм перевіряються та регулюються 54 рази за секунду.

- Раз на 1 секунду обчислюються показники А/ч і Вт/ч, час заряду, виводиться інформація на дисплей, перевіряється наявність та перевищення напруги від БП.
- Контролюється температура силового модуля; у разі необхідності вмикається вентилятор охолодження.
- Якщо температура перевищує встановлену межу (80 °С), струм заряду починає зменшуватися.
- Перевіряється справність силового транзистора.

Періодичність перевірок:

- Раз на 5 хвилин перевіряються умови завершення заряду.
- Раз на 10 хвилин А/ч, Вт/ч та час заряду зберігаються у пам'ять.

Управління:

- При натисканні на *Енкодер* або кнопку *Пуск* перемикаються екрани: чотири для дисплея 1602 та два для дисплея 2004. Через 7 секунд автоматично вмикається перший екран.
- Щоб увімкнути підсвітку, достатньо натиснути будь-яку кнопку або повернути *Енкодер*.
- Заряд можна перервати, утримуючи кнопку *Стоп* або *Енкодер*.

Примітка:

Якщо потрібно перервати заряд і продовжити пізніше, необхідно:

1. Вимкнути зарядний пристрій від мережі 220 В.
2. Від'єднати клеми від акумулятора.
3. Після повторного підключення до мережі та акумулятора заряд продовжиться.

Режим "Cycle" (Качелі)

Якщо ввімкнений режим "Cycle", при зниженні струму заряду до подвоєного значення струму завершення заряду ("Curr. charge min"), процес циклується:

- Струм заряду вимикається.
- Напруга на акумуляторі починає знижуватися.

- Коли зниження напруги становить менш ніж 0,01 В за секунду, знову вмикається струм заряду.

Це повторюється до закінчення часу заряду або до досягнення мінімального струму заряду.

Температурний контроль

Якщо підключений датчик температури акумулятора (NTC до піну A6 Arduino) і в параметрах увімкнено #define NTCTERM2 1, виконується контроль температури акумулятора.

- Для свинцево-кислотних акумуляторів ("Pb Ca/Ca", "Pb Ca+", "Pb Sur", "Pb AGM", "Pb Gel"):
  Виконується температурна корекція напруги заряду: +/-30 мВ на кожен градус Цельсія при відхиленні від +25 °C. Чим нижча температура акумулятора, тим вища напруга, і навпаки.
- Для літієвих та нікель-кадмієвих акумуляторів ("Li-ion", "LiFePo4", "NiCd/Mh"):
  - Якщо температура акумулятора менше 10 °C або більше 40 °C — заряд вимикається.
  - Якщо температура перевищує 35 °C — струм заряду зменшується на 10% за кожен градус.
  - Якщо температура менше 15 °C — струм заряду також зменшується на 10% за кожен градус.
- Для літій-титанатних акумуляторів ("LiTit"):
  - Якщо температура перевищує 40 °C, струм заряду зменшується на 10% за кожен градус.

Налаштування параметрів під час заряду

Напругу та струм заряду можна змінювати під час процесу. Для цього:

1. Перейдіть на четвертий екран (дисплей 1602) або на другий екран (дисплей 2004).
2. У першому рядку відображаються поточні напруга та струм.
3. Натисніть і утримуйте *Енкодер* або *Пуск* протягом 1 секунди.

4. У другому рядку з'являться встановлені межі напруги та струму.
5. Повертаючи *Енкодер* або кнопками *Плюс/Мінус*, змінюйте параметри.
6. Натискаючи *Енкодер* або *Пуск*, перемикайтеся між параметрами.
7. Щоб вийти із режиму налаштування, натисніть *Стоп* або утримуйте *Енкодер*.

Попередній заряд ("Precharge")

Призначений для зарядки сильно розряджених акумуляторів. Якщо цей режим увімкнений у налаштуваннях, заряд здійснюється малими струмами. Після закінчення заданого часу струм попереднього заряду поступово збільшується до встановленого струму заряду. На це може знадобитися до 64 годин. Після завершення попереднього заряду пристрій переходить у режим основного заряду.

Заряд - Дозаряд ("Charge-AddChar")

Спочатку здійснюється заряд, після чого активується режим дозаряду.

Заряд за методом Браніміра ("Charge Branim")

Цей режим передбачає заряд акумулятора малими струмами відповідно до інструкції Браніміра.

- Заряд відбувається малими струмами 1–2% від ємності акумулятора (наприклад, для акумулятора 60 А/год це становить 0,6–1,2 А).
- Коли напруга досягне напруги заряду, фіксується час, і через 20 годин вмикається заряд підвищеною напругою без обмеження струму, із силою струму 2–3% від ємності акумулятора (для 60 А/год це 1,2–1,8 А). Цей етап триває одну годину для перемішування електроліту.
- Після цього напруга знову обмежується до напруги заряду.

Цикл повторюється кожні 20 годин, доки струм заряду не знизиться до 0,1% від ємності акумулятора (для 60 А/год це 0,06 А).
Цей метод займає тривалий час (5–7 днів), але дозволяє розчинити застарілі сульфати.

Дозаряд ("Add charge")

Дозаряд працює в режимі "качелі".

- Кожні 10 секунд струм перемикається від мінімального до максимального, встановлених у налаштуваннях (струм дозаряду та мінімальний струм дозаряду).
- Це запобігає перегріву акумулятора та уникненню "закипання".

Розряд ("Discharge")

Розряд акумулятора здійснюється за допомогою лампи 12 В 55 Вт або потужного резистора 3 Ом 100 Вт струмом, встановленим у налаштуваннях.

- Струм розряду є імпульсним із ШІМ-регулюванням.
- Розряд завершується при зниженні напруги та струму до встановлених значень ("Volt dischar" — напруга розряду, "Curr.disch. Off" — струм завершення розряду).

Примітка:

- Якщо ви хочете, щоб розряд припинявся одразу після досягнення заданої напруги, встановіть значення "Curr.disch. Off" рівним "Curr.dischar" (струму розряду).

Під час розряду виконується обчислення:

- А/год, Вт/год.
- При досягненні 12 В розраховується поточна ємність акумулятора у відсотках.

Контрольно-тренувальний цикл (КТЦ) "Contr.trai.cycle"

Проводиться тренувальний цикл, який складається з наступних етапів:

1. Заряд.
2. Витримка (хвилини) — обчислюється як ємність акумулятора × 2 (наприклад, для акумулятора 60 А·год: 60 × 2 = 120 хвилин).
3. Розряд.
4. Заряд.
5. Якщо кількість циклів більше одного, КТЦ повторюється з пункту 2. Лічильник циклів зменшується на 1.
6. Дозаряд — виконується наприкінці КТЦ.

Цей цикл сприяє розчиненню застарілих сульфатів і покращенню стану акумулятора.

Залиті та злиті значення А·год і Вт·год для кожного циклу можна переглянути в розділі Статистика.

Зберігання ("Storage")

Після завершення зарядки, дозарядки або КТЦ активується режим зберігання. Відстежується напруга на акумуляторі, і при її зниженні до 13 В вмикається заряд малим струмом. Напруга підтримується на рівні 13,1 В для 12-вольтових акумуляторів.

Величину напруги можна задати в налаштуваннях.

Калібрування ("Calibration")

Виконується калібрування основних вимірюваних параметрів:

- "Volt/Curr akb":
    - Калібрування вимірювання струму під час заряду/розряду. Через те, що опір шунта, вказаний у скетчі, може бути неточним, необхідно встановити точне значення коефіцієнта для INA226, на основі якого розраховується струм.
    - Калібрування вимірювання напруги на акумуляторі під час заряду/розряду. Проводи зарядного пристрою та захисний транзистор мають опір, через який протікає струм, що спричиняє падіння напруги. Через це виміряна напруга на акумуляторі може бути некоректною, особливо при великих струмах. Для точного вимірювання напруги необхідно провести калібрування.

Послідовність дій для калібрування:

Калібрування струму та напруги акумулятора:

1. Підключіть клеми зарядного пристрою до акумулятора.
2. Підключіть мультиметр у режимі амперметра постійного струму послідовно до мінусової клеми акумулятора.
3. Зайдіть у розділ "Volt/Curr akb" у налаштуваннях зарядного пристрою.

- Якщо акумулятор не підключений, зарядник очікуватиме підключення. Для виходу натисніть кнопку Стоп або утримуйте Енкодер.

4. Після підключення акумулятора зарядник автоматично встановить струм розряду приблизно в 1 Ампер.

Калібрування струму:

5. Встановіть точний коефіцієнт для вимірювання струму:
   - Вертінням Енкодера або кнопками Плюс/Мінус змініть значення коефіцієнта так, щоб показання струму зарядника відповідали показанням амперметра.
6. Після калібрування струму:
   - Від’єднайте амперметр, перемкніть його в режим вимірювання напруги, та під’єднайте до клем акумулятора.
   - Мінусову клему зарядника також підключіть до акумулятора.

Калібрування напруги:

7. Короткочасно натисніть Енкодер або Пуск, щоб перейти до параметра калібрування напруги.
8. Змінюючи значення коефіцієнта, налаштуйте виміряну зарядником напругу так, щоб вона відповідала показанням мультиметра.
9. Після точного налаштування:
   - Натисніть кнопку Стоп або утримуйте Енкодер для завершення.
   - Калібрування буде завершено, а значення збережено в пам’ять.

Калібрування напруги блока живлення ("Volt Power"):

1. Перемкніть мультиметр у режим вимірювання постійної напруги та підключіть його до виводів блока живлення (БЖ).
2. Зайдіть у розділ "Volt Power" у налаштуваннях.
3. Змінюючи значення опорної напруги для АЦП Arduino, налаштуйте показання виміряної напруги БЖ зарядником так, щоб вони збігалися з показаннями мультиметра.
4. БЖ має бути підключеним до мережі 220 В під час калібрування.

ВИСНОВОК

У ході виконання магістерської роботи на тему «Методи і засоби програмної підтримки вбудованої комп’ютеризованої системи для універсального зарядного пристрою» було досягнуто поставлених цілей та розв’язано низку технічних і програмних завдань, спрямованих на створення ефективного, адаптивного та універсального зарядного пристрою. Робота охоплювала як теоретичний, так і практичний аспекти розробки, що дозволило отримати повноцінний робочий прототип із широкими функціональними можливостями.

Розробка апаратної частини

У процесі виконання магістерської роботи було створено електричну схему зарядного пристрою, яка враховує сучасні вимоги до енергоефективності, безпеки та гнучкості використання. Основою пристрою є мікроконтролер ATmega328P, інтегрований у платформу Arduino Nano, яка забезпечує високу функціональність, доступність і простоту інтеграції з іншими компонентами.

Основні компоненти апаратної частини

1. Мікроконтролер ATmega328P
   Виконує центральну обчислювальну роль у пристрої, забезпечуючи обробку даних з датчиків, управління зарядним процесом і виконання алгоритмів заряджання.
2. Датчики напруги та струму
   Датчики забезпечують точний контроль за станом акумулятора під час заряджання. Зокрема, вони фіксують поточну напругу на клемах батареї та струм заряджання,. Для цього використовувалися недорогі, але надійні сенсори на базі INA226.
3. MOSFET транзистори
   Виступають у ролі силових елементів для управління струмом заряджання. Їх використання забезпечує низькі втрати енергії, швидке перемикання та стабільну роботу пристрою навіть при високих струмах. Крім того, завдяки високій надійності MOSFET транзисторів забезпечується довговічність зарядного пристрою.

4. Дисплей

   Для відображення параметрів роботи пристрою, таких як напруга, струм, залишковий час заряджання або повідомлення про помилки, було використано LCD-дисплей 1602 з I2C адаптером. Це дозволило зменшити кількість підключених проводів і спростити схему.

5. Джерело живлення та модулі захисту

   Апаратна частина зарядного пристрою включає модуль стабілізації напруги та фільтри для усунення перешкод, а також додаткові модулі захисту:

   - Від перенапруги.
   - Від перевантаження за струмом.
   - Від перегріву пристрою.
   - Від зворотної полярності підключення акумулятора.

6. Резистивно-ємнісні компоненти

   Забезпечують стабільність роботи схеми, зменшують пульсації та захищають від перешкод у мережі. Їх оптимальний підбір був важливим для досягнення високої точності вимірювань.

Особливості реалізації апаратної частини

Важливою перевагою розробленого пристрою є його модульна структура, що дозволяє легко замінювати або додавати нові компоненти. Наприклад, можна інтегрувати бездротовий модуль зв'язку для віддаленого моніторингу або оновити датчики для роботи з іншими типами батарей. Системний підхід до розробки апаратної частини дозволив створити комп’ютеризований зарядний пристрій, що відповідає сучасним вимогам до безпеки, енергоефективності та адаптивності.

Програмна реалізація

Основна увага при розробці методів і засоби програмної підтримки вбудованої комп’ютеризованої системи для універсального зарядного пристрою»була приділена створенню ефективної та адаптивної прошивки для мікроконтролера ATmega328P, яка забезпечує виконання інтелектуальних алгоритмів заряджання.

Використовуючи мову програмування C++ та інтегроване середовище розробки Arduino IDE, вдалося реалізувати функціонал, що дозволяє гнучко налаштовувати процес заряджання під різні типи акумуляторів, забезпечуючи їхню безпеку та енергоефективність.

Основні функції програмного забезпечення

1. Адаптивне заряджання з багатофакторним контролем
   Програма дозволяє забезпечити точний контроль над такими параметрами, як:
   - Напруга на акумуляторі.
   - Струм заряджання.
   - Температура акумулятора та силових елементів.
2. Режими роботи
   Прошивка підтримує кілька режимів роботи, які користувач може налаштовувати або обираються на основі аналізу параметрів акумулятора:
3. Інтерфейс користувача
   Для забезпечення зручності роботи було реалізовано:
   - Виведення поточних параметрів заряджання (напруга, струм, стан заряджання) на LCD-дисплей.
   - Індикацію стану пристрою за допомогою LED-індикаторів (наприклад, завершення заряджання чи повідомлення про помилку).
   - Сповіщення про можливі помилки, такі як перегрів, перевантаження або неправильне підключення акумулятора.
4. Додаткові функції захисту
   Реалізовано програмні механізми захисту, що включають:
   - Відключення заряджання при перевищенні встановленої температури.
   - Захист від короткого замикання.
   - Виявлення розриву ланцюга або неправильного підключення батареї.

Інструменти та засоби програмування

Для розробки та тестування програмного забезпечення використовувалися наступні інструменти: Arduino IDE: забезпечує зручний процес написання, компіляції та завантаження коду на мікроконтролер..

Бібліотеки Arduino: стандартні та сторонні бібліотеки допомогли спростити роботу з периферійними пристроями, такими як дисплей, датчики та модулі захисту.

Результати програмної реалізації

Створена прошивка дозволила досягти високого рівня адаптивності зарядного пристрою до різних умов роботи та типів акумуляторів. Вона забезпечує:

Ефективне використання ресурсів мікроконтролера.

Надійність і стабільність роботи завдяки продуманій структурі коду.

Простоту у модифікації та доповненні новими функціями, такими як бездротовий моніторинг чи оновлення програмного забезпечення.

Розроблене програмне забезпечення у поєднанні з апаратною частиною утворює надійний і гнучкий зарядний пристрій, що відповідає сучасним вимогам енергоефективності, безпеки та користувацького комфорту.

Використання програмних інструментів

У процесі роботи над проектом були використані сучасні програмні інструменти, які забезпечили ефективне проектування, розробку та тестування як апаратної, так і програмної частини зарядного пристрою. Це дозволило оптимізувати всі етапи роботи — від створення електричної схеми до налаштування мікроконтролера.

KiCad EDA

Для створення електричної схеми та розробки друкованої плати використовувалася KiCad EDA — потужна та багатофункціональна система автоматизованого проектування. Використання цього інструменту дозволило:

- Розробити електричну схему пристрою з урахуванням усіх необхідних компонентів, включаючи мікроконтролер, датчики, силові елементи та модулі захисту.

- Створити макет друкованої плати (PCB), оптимізувавши розташування компонентів для мінімізації електромагнітних перешкод і забезпечення надійності роботи.

Arduino IDE

Для написання, компіляції та завантаження прошивки в мікроконтролер використовувалася Arduino IDE. Цей інструмент надав розробнику простий у використанні інтерфейс та широкий спектр функцій для розробки програмного забезпечення. Завдяки Arduino IDE були виконані наступні завдання:

- Написання коду на мові програмування C++ для реалізації алгоритмів заряджання та управління пристроєм.
- Інтеграція бібліотек для роботи з периферійними пристроями, такими як датчики напруги та струму, дисплей і модулі захисту.
- Завантаження прошивки в мікроконтролер через USB-інтерфейс, що спростило процес налаштування та перевірки роботи пристрою.

Arduino IDE забезпечила високу гнучкість у створенні та модифікації коду. Завдяки використанню численних бібліотек, розробка функціоналу була значно пришвидшена, а підтримка мови C++ дозволила ефективно організувати програмну логіку.

Перспективи розвитку

Робота заклала міцний фундамент для подальшого вдосконалення методів та засобі програмної підтрики комп’ютеризованого універсального зарядного пристрою, відкриваючи перспективи для розширення його функціональності та підвищення ефективності. Основними напрямками розвитку є:

1. Додавання бездротового модуля
2. Інтеграція більш точних сенсорів
3. Оптимізація алгоритмів заряджання
4. Розширення функціональності для користувача
5. Модернізація апаратної частини.

6. Інтеграція з альтернативними джерелами енергії
   Можливість адаптації пристрою для роботи від сонячних панелей або інших відновлюваних джерел енергії забезпечить екологічність та енергоефективність.

У ході роботи було розроблено комп'ютеризований універсальний зарядний пристрій, який поєднує простоту апаратної реалізації з гнучкістю програмних рішень. Основою проєкту став мікроконтролер ATmega328P, інтегрований у платформу Arduino Nano, що забезпечило доступність, універсальність і надійність системи. У рамках розробки були виконані такі завдання:

- Створено електричну схему пристрою за допомогою KiCad EDA, що дозволило врахувати всі вимоги до апаратної частини.
- Розроблено методи і засоби програмної підтримки (прошивку) на мові програмування C++ у середовищі Arduino IDE, що реалізує інтелектуальні алгоритми заряджання.
- Виконано тестування пристрою, перевірено його стабільність і відповідність заданим технічним параметрам.

Реалізація проєкту дала змогу на практиці застосувати теоретичні знання в галузі електроніки, схемотехніки та програмування, а також поглибити навички роботи з сучасними інструментами автоматизованого проектування. Зарядний пристрій продемонстрував високу функціональність, можливість адаптації до різних типів акумуляторів та перспективність використання в реальних умовах. Отримані результати підтвердили доцільність використання мікроконтролерів для створення програмно-апаратних систем із гнучким налаштуванням та високим рівнем інтеграції.

Подальший розвиток методів та засобів підтримки програмної підтримки комп'ютеризованої системи зарядного пристрою відкриває перспективи для вдосконалення функціональних можливостей, забезпечуючи його адаптацію до майбутніх вимог ринку

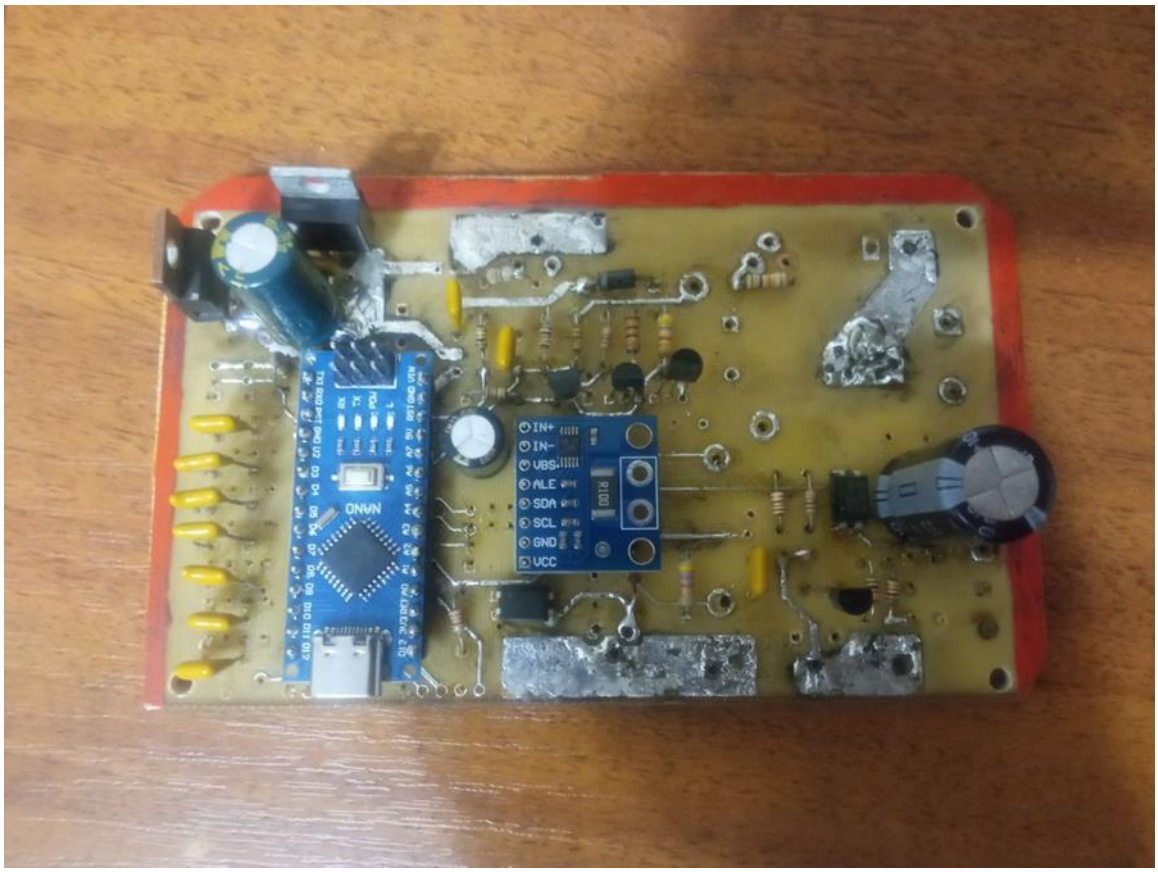
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arduino IDE Documentation [Електронний ресурс]. – Режим доступу: https://docs.arduino.cc .
2. ATmega328P Datasheet, Microchip Technology Inc., 2020.
3. Bart Van Damme. Power Supply Design for Engineers. Elsevier, 2021. 340 p.
4. Bishop, Owen. Programming Microcontrollers in C. Newnes, 2007.
5. Dhananjay Gadre. Programming and Customizing the AVR Microcontroller. McGraw-Hill, 2001.
6. Development of Charging Algorithms for Li-ion Batteries. IEEE Transactions on Power Electronics, 2022. Vol. 37, No. 5. P. 4300–4312.
7. Енергозберігаючі технології в роботі зарядних пристроїв [Електронний ресурс]. – Режим доступу: https://www.energy.gov
8. Gajski D.D. Embedded Systems Foundations of Cyber-Physical Systems. 3rd edition. Springer, 2021. 512 p.
9. INA226 Current Sensor Datasheet [Електронний ресурс]. – Режим доступу: https://www.ti.com/product/INA226?keyMatch=ina226&tisearch=universal_search
10. KiCad EDA User Manual [Електронний ресурс]. – Режим доступу: https://kicad.org
11. LiFePO4 Battery Technologies: State of the Art and Future Trends [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/333112345
12. Mazidi, Muhammad Ali. The AVR Microcontroller and Embedded Systems: Using Assembly and C. Pearson Education, 2011.
13. Minch, B. "Battery Charging Technology for Portable Devices," IEEE Transactions on Power Electronics, Vol. 29, No. 8, 2014, pp. 4579–4591.
14. NXP Semiconductors. Designing Safe Embedded Systems. Application Note [Електронний ресурс]. – Режим доступу: https://www.nxp.com
15. Open Source Hardware Association. Best Practices for Embedded System Design [Електронний ресурс]. – Режим доступу: https://www.oshwa.org

16. Owen Bishop. Programming Microcontrollers in C. Newnes, 2007.
17. Pop, V., Bergveld, H. J., Notten, P. H., and Regtien, P. P. L. Battery Management Systems: Design by Modelling. Springer, 2008.
18. Proakis J.G., Manolakis D.G. Digital Signal Processing: Principles, Algorithms, and Applications. 5th edition. Pearson Education, 2020. 992 p.
19. Rashid, M.H., "Battery Chargers for Renewable Energy Applications: A Review," Renewable and Sustainable Energy Reviews, Vol. 39, 2014, pp. 131–139.
20. Smith R. Embedded Systems Design. 4th edition. New York: Wiley, 2020. 512 p.
21. STM32CubeMX Documentation [Електронний ресурс]. – Режим доступу: https://www.st.com/en/development-tools/stm32cubemx.html
22. Технології бездротового зв’язку Wi-Fi для вбудованих систем [Електронний ресурс]. – Режим доступу: https://www.wifi.org
23. Український інститут стандартизації. Рекомендації зі створення друкованих плат. Київ: УкрНДІСТ, 2020.
24. Шкадовський М.І. Алгоритми керування заряджанням акумуляторів: огляд сучасних підходів. Наукові записки кафедри електроніки, 2019. Т. 12, №2. С. 45–58.

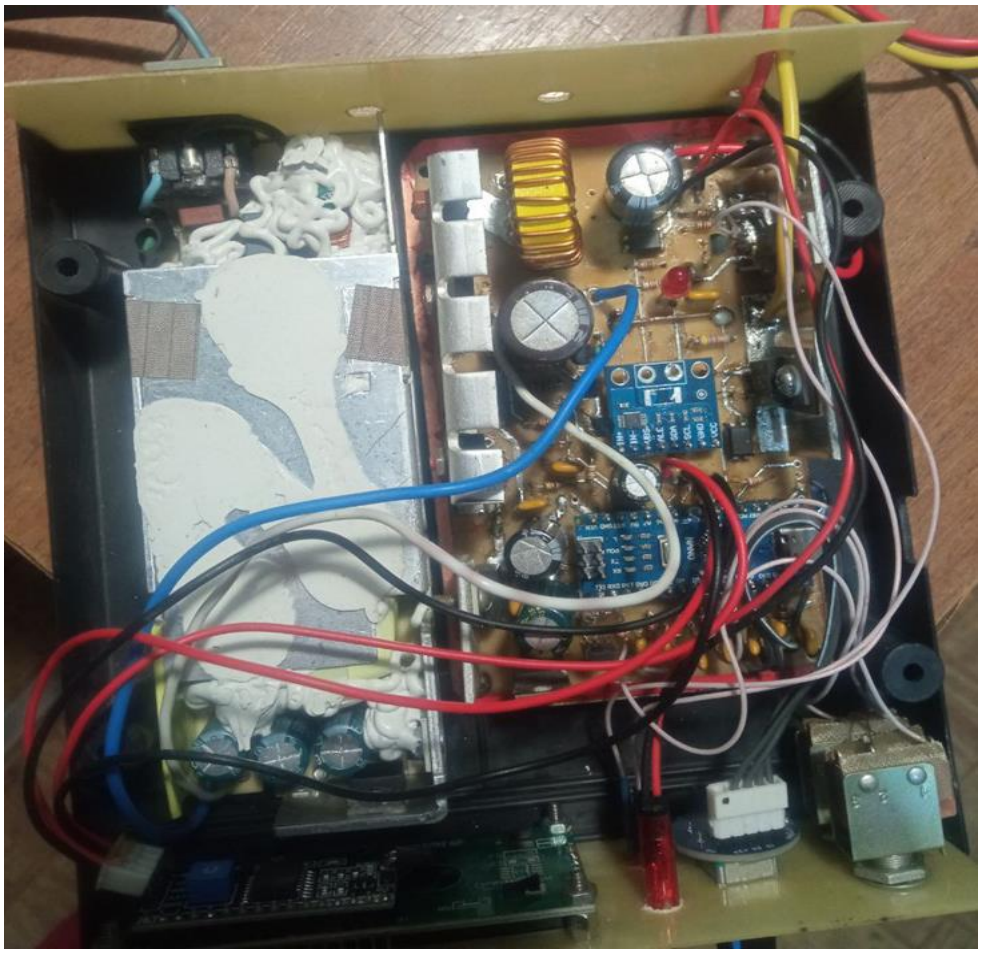
ДОДАТОК А

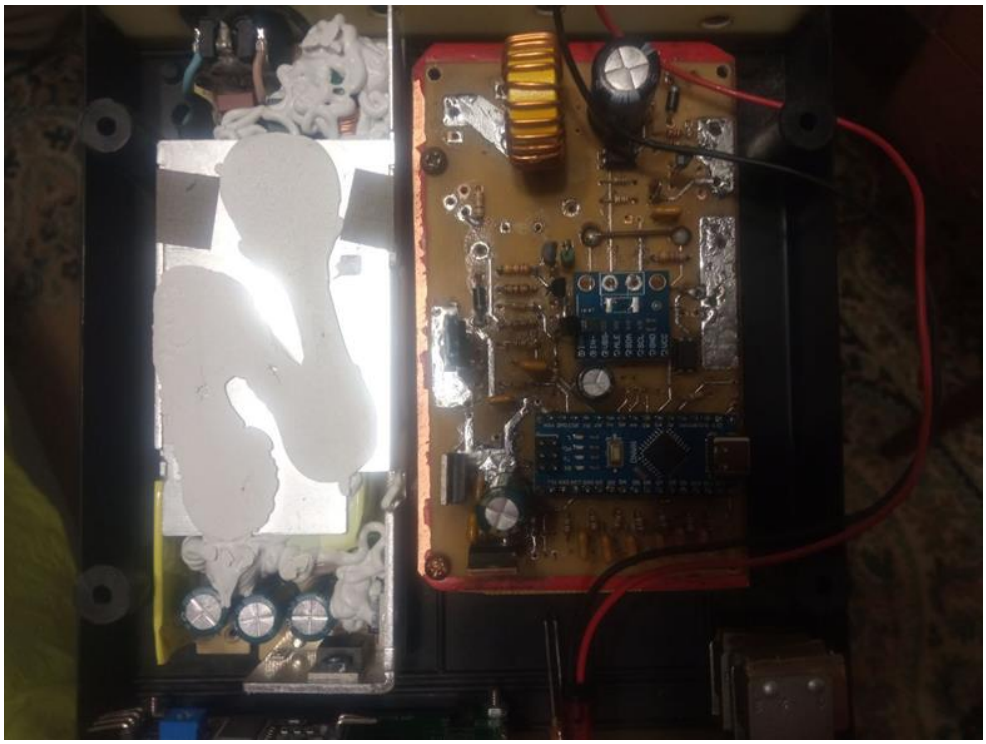
Рис. А.1. Вигляд Реалізованої плати та компонентів Комп’ютеризованої зярядної системи



ДОДАТОК Б

Рис.Б.2 Зовнішній вигляд внутрішніх складових комп’ютеризованої системи



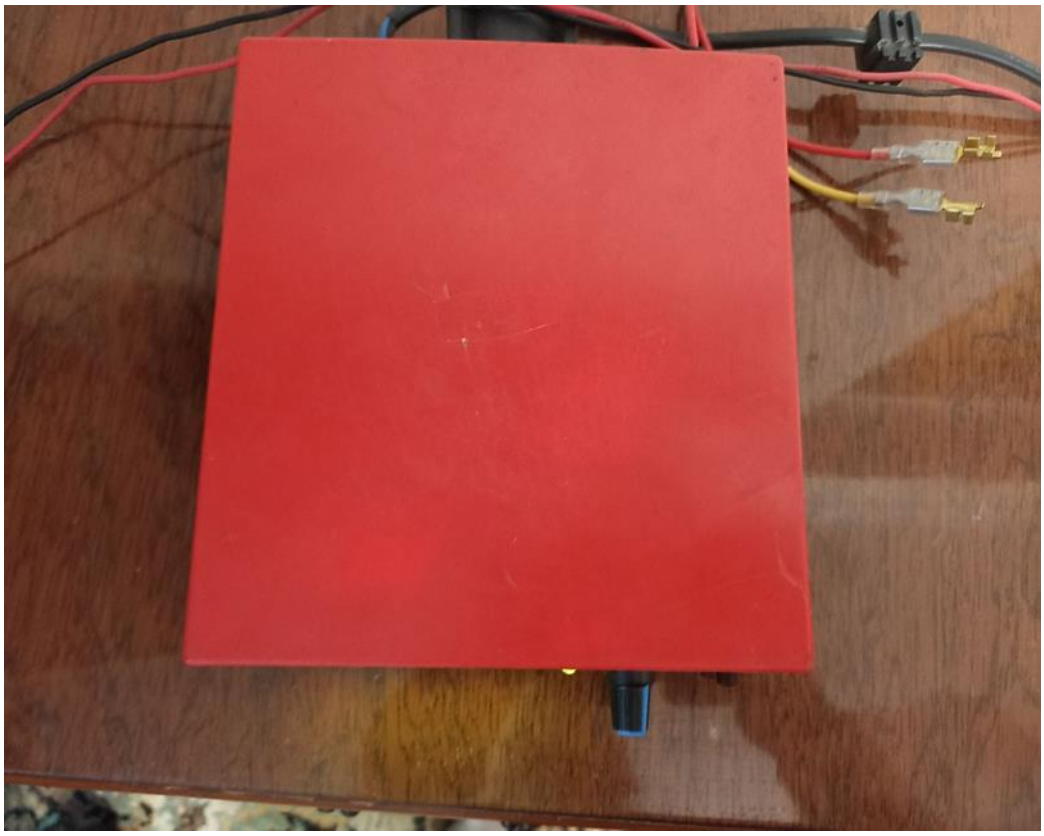


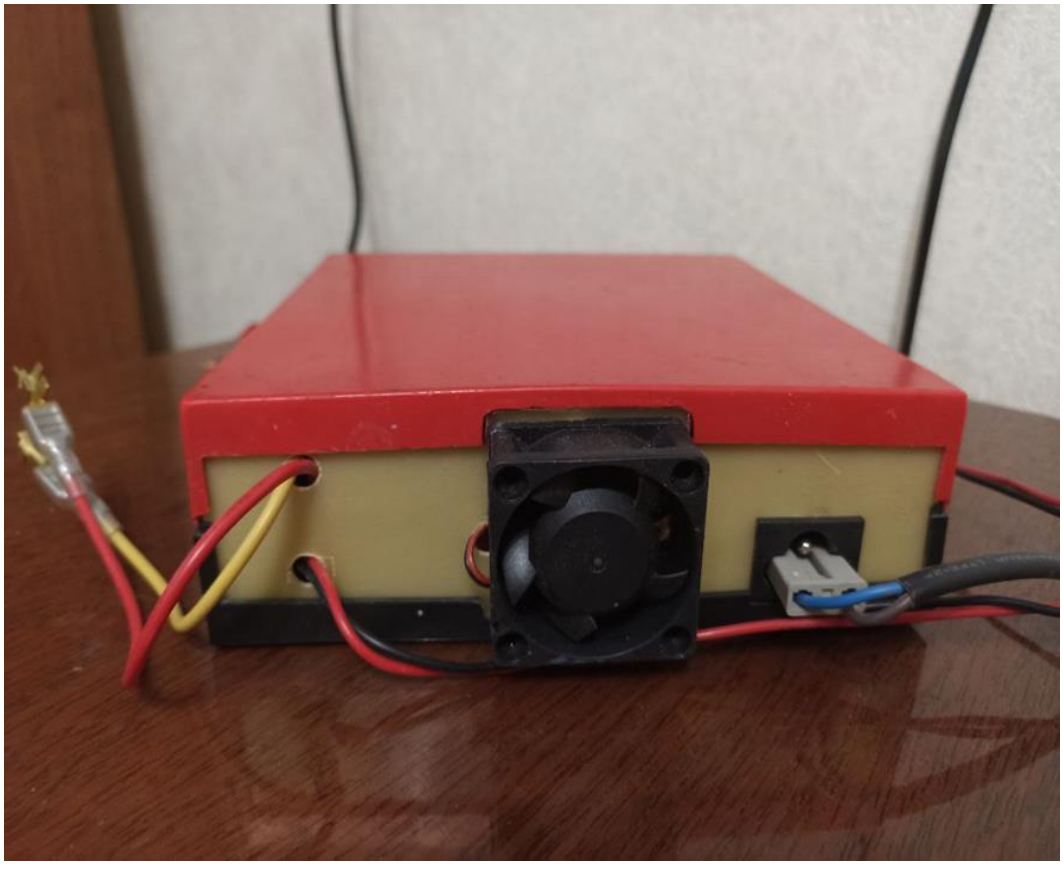
ДОДАТОК В

Рис.В.3 Зовнішній вигляд фронтальної панелі корпусу пристрою

ДОДАТОК Г

Рис.Г.4 Зовнішній вигляд корпусу комп’ютеризованої системи зверху та ззаду





ДОДАТОК Д

Таблиця Д.1 Параметри основних компонентів комп’ютеризованої системи

Параметри	Бібліотека	Бібліотека Ref
Arduino_Nano_v3.x	MCU_Module	Arduino_Nano_v3.x
INA226		
0,1	мФ	керамічний
0,1	мФ	керамічний
0,1	мФ	керамічний
0,1	мФ	керамічний
0,1	мФ	керамічний
0,1	мФ	керамічний
0,1	мФ	керамічний
1000 35В	мФ	електролітичний
0,1	мФ	керамічний
1000 35В	мФ	електролітичний
0,1	мФ	керамічний
3000 35В	мФ	електролітичний
0,1	мФ	керамічний
1000 25В	мФ	електролітичний
0,1	мФ	керамічний
470мкФ 6В	мФ	електролітичний
0,1 250В	мФ	керамічний
0,1	мФ	керамічний
MBR1645	Діод Шотки на ток від 10А.	MBR1645_TO220
1N4007	діод	1N4007
LED	Світлодіод 2,5В	LED
LED	Світлодіод 2,5В	LED
1N4007	діод	1N4007
LED	Світлодіод 2,5В	LED
LED	Світлодіод 2,5В	LED
0,5 А	0,5 А	Запобіжник,перемекач
0,5 А	0,5 А	Запобіжник,перемекач
0,5 А	0,5 А	Запобіжник,перемекач
8А	8А	Запобіжник,перемекач
0,04мГ	МіллиГенрі	Дросель
LSD16x2_I2C	lsd16x2_i2c	LSD16x2_I2C
Спікер	Динамік_Кристал	Пищалка
IRF4905	Транзистор_FET	IRF4905
2SC1815	Транзистор_BJT	2SC1815
2SC1815	Транзистор_BJT	2SC1815
IRF3205	Транзистор_FET	IRF3205
IRF3205	Транзистор_FET	IRF3205
IRF3205	Транзистор_FET	IRF3205
2SC1815	Транзистор_BJT	2SC1815 npn
2SA1015	Транзистор_BJT	2SA1015 pnp
BT136-BT139	Сімістор_Тиристор	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	

1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
4,7 тис	Резистор 0,125Вт	
10 тис	Резистор 0,125Вт	
200 Ом	Резистор 0,125Вт	
10 Ом	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
1K	Резистор 0,125Вт	
200 Ом	Резистор 0,125Вт	
10 тис	Резистор 0,125Вт	
4K7	Резистор 0,125Вт	
0,01 — 0,004	Шунт, Ом.	0,081/0,01=8,1A; 0,081/0,004=20,25A
1 тис	Резистор 0,125Вт	
4K7	Резистор 0,125Вт	
300 Ом	Резистор 0,125Вт	
4K7	Резистор 0,125Вт	
100 Ом	Резистор 0,125Вт	
10 тис	Резистор 0,125Вт	
300 Ом	Резистор 0,125Вт	
330 Ом	Резистор 0,25Вт	
330 Ом	Резистор 0,25Вт	
68 Ом	Резистор 0,125Вт	
10 тис	Резистор 0,125Вт	
1 тис	Резистор 0,125Вт	
Старт	Кнопка	
+>	Кнопка	
<-	Кнопка	
Стоп	Кнопка	
Кодувальник	Енкодер	
SW_SPST	Виключатель	
LM7812	Лінійний стабілізатор 12В	LM7812_TO220
PC817	Оптопара	PC817
DS18B20	Датчик температури	DS18B20
MOC3041M	Оптопара	MOC3041M
PC817	Ізолятор	PC817
LM7805	Лінійний стабілізатор 5В	LM7805_TO220

ДОДАТОК Е

Таблиця Е.1 Перерахунок основних компонентів запчастин

Резистори			
1	**кОм**	**11**	**шт**
4,7	**кОм**	**4**	**шт**
10	**кОм**	**4**	**шт**
10	**Ом**	**1**	**шт**
68	**Ом**	**1**	**шт**
100	**Ом**	**1**	**шт**
200	**Ом**	**2**	**шт**
300	**Ом**	**1**	**шт**
330	**Ом**	**2**	**шт**
0,01-0,004	**Ом**	**1**	**шт**
Конденсатори			
0,1	**мФ**	**12**	**шт**
0,1	**мФ 220В**	**1**	**шт**
1000	**мФ 35В**	**3**	**шт**
3000	**мФ**	**1**	**шт**
470	**мФ 6В**	**1**	**шт**
Транзистори			
IRF4905		**1**	**шт**
IRF3205		**3**	**шт**
2SC1815		**1**	**шт**
2SA1015		**1**	**шт**
			шт
Стабілізатори			**шт**
LM7805		**1**	**шт**
LM7812		**1**	**шт**
Діоди			
MBR1645		**1**	**шт**
1N4007		**2**	**шт**
Оптопара			
PC817		**2**	**шт**
MOC3041M		**1**	**шт**
Тиристор			
BT136		**1**	**шт**
Енкодер		**1**	**шт**
кнопки		**2-4**	**шт**
Блок живлення від ноутбука		**1**	**шт**
DS18B20		**1-2**	**шт**
LSD16x2_I2C	**Дисплей з платою I2C**	**1**	**шт**

Плата Arduino Nano V3	**Плата Arduino Nano V3**	**1**	**шт**
Пищалка		**1**	**шт**
INA226		**1**	**шт**

ДОДАТОК Ж

Програмний код для комп'ютеризованої системи універсального зарядного пристрою

```
Реалізація розділу основного Налаштування користувача
#define MEMVER 102 // версія прошивки в EPPROM
#if defined(__LGT8FX8P__)
#define VER "Ver 1.02-lgt8" // версія прошивки та тип контролера
#else
#define VER "Ver 1.02-328p" // версія прошивки та тип контролера
#endif
#define INTERFACE 1 // вибір мови інтерфейсу:  1 - англійська, 2 - трансліт
// параметри дисплея
#define ADDRDISP 0x27 // адреса дисплея 0x27 або 0x3F
#define DISPLAYx 16 // кількість стовпців дисплея 16 або 20
#define DISPLAYy 2 // кількість рядків дисплея 2 або 4
// Параметри управління
#define ENCBUTT 2 // 0 - тільки енкодер, 1 - тільки кнопки, 2 - енкодер + Пуск + Stop, 3 - енкодер + всі кнопки
// Тип спікера
#define SPEAKER 1 // 1 - пасивний спікер, 2 - активний спікер
// Вибір програмних модулів
#define VOLTIN 1 // замір напруги від БП: 1 – використовується, 0 – не використовується
#define FAN 1 // управління вентилятором: 0 - не використовується, 1 - вентилятор с плавним регулюванням оборотів ШІМ, 2 - режим роботи вентилятора вкл/вимк.
#define PROT 1 // управління модулем захисту: 1 - використовується, 0 - не використовується
#define DISCHAR 1 // розряд: 1 – використовується розрядний модуль, 0 – не використовується.
#define RESIST 1 // вимір опору: 1 - використовується, 0 - не використовується
#define BRANIMIR 1 // методика заряду Акб за інструкцією Браніміра: 1 - використовується, 0 - не використовується
// реле 220В або світлодіод БП для пін 10
#define POWPIN 0 // 0 - не використовується, 1 - реле 220 Вольт, 2 - індикатор включення БП до мережі (не працює без #define VOLTIN 1).

//>>> при зміні інших параметрів потрібно зробити "Скидання системних налаштувань", щоб дані параметри застосувалися і збереглися в пам'ять.
// Опір шунта
#define SHUNT 0.01085 // опір шунта, Ом
#define CURR_MAX (0.08192f/(float)SHUNT) // Максимальний струм зарядника, 25.5 ампер. розраховується автом.(8.1А при шунті 0,01 Ом). Можна виставити вручну. Максимальний струм залежить від опору шунта (0.08192 / SHUNT) 0.08192V / 0.01 = 8.192A (Макс напр на шунті / сопрот шунта)
//#define CURR_MAX 8.0// Максимальний струм ЗУ 8.0 Ампер. При необхідності розкоментуй рядок, впиши максимальний струм вручну. Рядок вище закоментуй.

#define OHMS 61 // опір лінії до акб - мілліОм, 0-255. Встанови 0 якщо хочеш щоб INA заміряла напругу без корекції струму.
// Щоб значення застосували скидання калібрування.

// Параметри для функції виміру напруги від БП. Опорна напруга та значення опорів дільника напруги.
#define VREF 5000 // точне напруження на піні 5V (мВ). Виміряй напругу і впиши сюди значення. Впливає на вимірювання напруги від БП. 4292 4304
#define DIV_R1 10000 // Точне значення верхнього резистора Ом. до 65кОм. До 35В від БП рекомендується на 6000 Ом. до 55В – 10000 Ом.
#define DIV_R2 1000 // Точне значення нижнього резистора Ом.

// Вибір датчика температури 1 контролю температури силового транзистора.
#define SENSTEMP1 1 // датчик температури транзистора Q1: 1 – DS18B20, 2 – NTC, 0 – не використовується. Схема 1.6.3
// Параметри для датчика температури NTC
#define NTCB1 3435 // В термістора - береться з характеристик термістора
#define NTCR1 10000 // Ом – опір термістора при 25 градусах.
#define NTCRS1 10000 // Ом - опір підтягуючого резистора (тим точніше краще)

// датчик температури 2 контролю температури акумулятора.
#define SENSTEMP2 0 // датчик температури акумулятора: (1 - DS18B20 поки немає), 2 - NTC, 0 - не використовується. Схема 1.6.3
// Параметри для датчика температури NTC
#define NTCB2 3435 // В термістора - береться з характеристик термістора
#define NTCR2 10000 // Ом – опір термістора при 25 градусах.
#define NTCRS2 10000 // Ом - опір підтягуючого резистора (тим точніше краще)

// Параметри управління вентилятором
#define DEG_ON 40 // температура включення вентилятора у градусах
#define DEG_OFF 36 // температура відключення вентилятора у градусах
```

#define DEG_MAX 60 // температура максимальних обертів вентилятора
#define DEG_CUR 70 // температура при якій зменшується струм заряду
#define CURFAN_ON 1500 // значення струму в міліамперах при якому включається вентилятор. (До 32А)
#define CURFAN_OFF 1200 // значення струму в міліамперах при якому відключається вентилятор. (До 32А)
#define CURFAN_MAX 7000 // значення струму в міліамперах для максимальних обертів вентилятора. (До 32А)

// Значення напруги БП
#define POWER 24 // 24 - напруга від блока живлення, вольт(ціле число). Якщо встановлено дільник напруги на вході, то заміряється в програмі. (Цифрове обмеження 65 Вольт)
#define POWER_MAX 26 // максимальна допустима напруга від БП (17-32 Вольт). Якщо воно перевищить, то відключиться реле 220В (пін 10) на 10 секунд (якщо був просто стрибок напруги). Утитуй напругу С12 та С10 та стабілізатора LM7812.
// час дисплея
#define TIME_LIGHT 3 // час підсвічування дисплея у хвилинах
#define TIME_DISP 10 // час повернення на 1 екран на секундах
// профілі
#define PROFIL 9 // кількість профілів користувача 0 – 9.

// зчитування температури з датчика SENSTEMP1 при розряді та керування кулером
#define KULDISCHAR 1 // 0 - вимкнути. 1 - керування кулером по датчику температури SENSTEMP1. 2 – примусове включення вентилятора при розряді
/ / Графік напруги і струму при заряді за останню годину.
#if defined(__LGT8FX8P__)
#define GRAFIK 0// 1 - вкл, 0 - вимк. Дані фіксуються кожні 5 хвилин. У LGT8 не влазить.
#else
#define GRAFIK 1// 1 - вкл, 0 - вимк. Дані фіксуються кожні 5 хвилин. У LGT8 не влазить.
#endif

// Зміна сервісних параметрів
#define SERVICE 1// 1 - вкл, 0 - вимк.
// Зменшення струму заряду при перевищенні температури силового транзистора.
#define GUARDTEMP 1// 1 - вкл, 0 - вимк.
// Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт. Включення пін А0.
#define GUARDA0 0// 1 - вкл, 0 - вимк.
// ЦАП MCP4725
#define MCP4725DAC 0 // 1 - виведення значень заряду в ЦАП MCP4725, 0 - вимкнено
#define ADDR4725 0x60 // адреса MCP4725
// Логгер. Відправлення до мережного порту.
#define LOGGER 0 // функція відправлення значень мережного порту (швидкість 115200). Займає багато пам'яті. Якщо хочеш її використати, відключи інші функції.
// 1 - набір логів: час, напруга, струм, А/год, Вт/год, темп Акб, темп Q1, напруга БП.
// 2 – набір логів: час, напруга, струм.
// 0 – не використовується.
#define LOGGTIME 1 // Період, в секундах, 1 - 255. З цим періодом будуть відправлятися дані в мережевий порт.
// Налаштування пінів на власний розсуд. Назву залишай номер піна міняй.
#define PUSK 2 // кнопка - Пуск
#define PWMCH 3 // ШІМ вихід заряду (3 вихід ШІМ)
#define MINUS 4 // кнопка - Мінус
#define PLUS 5 // кнопка - Плюс //(5 вихід ШИМ)
#define STOP 6 // кнопка - Стоп //(6 вихід ШИМ)
#define ENC_S1 7 // енкодер S1
#define ENC_S2 8 // енкодер S2
#define PWMKUL 9 // ШІМ - вентилятор (кулер) (9 вихід ШІМ)
#define RELAY220 10 // управління реле 220В //(10 вихід ШІМ)
#define PWMDCH 11 // ШИМ навантаження. Розряд. (11 вихід ШІМ)
#define ENC_KL 12 // енкодер klick
#define PIN_13 13 // управління реле відключення навантаження від акб у буферному режимі
#define PINA0 A0 // управління блокуванням модуля захисту
#define BUZER A1 // Вихорд на динамік (піщалка). При вкл. генеруються імпульси із частотою близько 1200Гц. Можна використовувати піщалку від комп'ютера підключивши безпосередньо до піна та GND.
#define PROTECT A2 // Управління захисним транзистором Q4
#define PINTERM1 A3 // вхід датчика температури DS18B20 або NTC
#define PINTERM2 A6 // вхід датчика температури NTC
#define POWIN A7 // замір напруги БП
// A4 // SDA
// A5 // SCL

// ------- кінець налаштувань --------

```
// ****можна змінити за бажання***
// Мінімальний струм завершення заряду
#define CURMIN 20
// Мінімальний струм заряду, ма
#define CUR_CHARGE_MIN 100
// крок зміни напруги в налаштуваннях, мВ
#define STEP_VOLT 100
// крок зміни струму в налаштуваннях, ма
#define STEP_CURR 100

/* Цифрові контакти:
 D2 - кнопка1 ОК/Старт
 D3 - ШИМ вихід заряду
 D4 - кнопка мінус-
 D5 – кнопка плюс+
 D6 - Стоп/Назад
 D7 - енкодер S1
 D8 - енкодер S2
 D9 - ШИМ - вентилятор (кулер)
 D10 - управління реле 220В або світлодіод, що відображає наявність живлення
 D11 - ШИМ навантаження. Розряд.
 D12 - енкодер klick
 D13 - управління реле відключення навантаження від акб у буферному режимі

 Аналогові контакти:
 A0 - управління блокуванням модуля захисту
 A1 - вихід звук
 A2 - вихід керування захистом
 A3 – вхід датчика температури силового транзистора
 A4 - SDA
 A5 - SDL
 A6 – вхід датчика температури NTC акумулятора
 A7 – замір напруги на вході зарядника
*/

/*
 const char* TypeAkb[] = {
 "Pb Ca/Ca", // 0 2,1 напр. напр. відсічення при розряді - 1,75В
 "Pb Ca+", // 1
 "Pb Sur", // 2
 "Pb AGM", // 3
 "Pb Gel", // 4
 "Li-ion" // 5 літій-іонні 3,65В напр. відпр. при заряді - 4,1 напр. відсічення при розряді - 2,5-3В (2,75В) струм заряду - 1С напр зберігання - 3,75
 "LiFePo4", // 6 літій-залізо-фосфатні 3,2В напр. відпр. при заряді - 3,3 напр. відсічення при розряді - 2,8В(2В) напр зберігання - 3,3 струм заряду - <4С
 "LiTit", // 7 літій-титанатні 2,4В
 "NiCd/Mh", // 8 нікель-кадмієві/металгідридні 1,25В/1,37В струм заряду - 1С напр. відсічення при заряді - 1,6В(1,37В) (0,85В/1В)
 };
 // Li-Pol літій-полімерний 3,7В напр. відсічення при заряді - 4,2 напр зберігання - 3,85 напр. відсічення при розряді - 3В
*/
//******** НЕ торкатися **********
#if defined(__LGT8FX8P__)
#define BITRATE 4096 // розрядність АЦП
#define BIT 12 // розрядність АЦП біт
#else
#define BITRATE 1024 // розрядність АЦП
#define BIT 10 // розрядність АЦП біт
#endif

// частота силового модуля {0.25, 0.5, 1, 2, 4, 8, 31, 62} кГц
#define FREQ_CHARGE 7
// Частота розрядного модуля {0.25, 0.5, 1, 2, 4, 8, 31, 62} кГц
#define FREQ_DISCHARGE 4
// Коефіцієнт напруги INA226 в мікровольтах + 1
#define INAVOLTKOOF 250
// адреса початкового осередку пам'яті для pam
#define MEM 30
```

```
// адреса початкової комірки пам'яті результатів виміру ємності КТЦ. Максимум 10 вимірів по 16 байт
#define MEM_KTC 850
// Налаштування команд енкодера та кнопок
#define ENCLEFT -1
#define ENCRIGHT 1
#define ENCCLICK 16
#define ENCHELD 2
#define ENCSTEP 4
#define OKCLICK 26
#define OKHELD 12
#define STOPCLICK 36
#define STOPHELD 22
#define PLUSCLICK 46
#define PLUSHELD 32
#define PLUSSTEP 34
#define MINUSCLICK 56
#define MINUSHELD 42
#define MINUSSTEP 44


// Назва біт для pam.MyFlag
#define CHARGE 0
#define OSCIL 1
#define BRANIM 2
#define PRECHAR 3
#define BRANIMBOIL 4 // прапор перемішування електроліту
#define FLAG5 5
#define FLAG7 6
#define FLAG8 7

// Назва біт для vkr.GlobFlag
#define TRQ1 0
#define INAERR 1
#define FL2 2
#define FL3 3
#define FL4 4
#define FL5 5
#define FL6 6
#define FL7 7

#define CHARGb 0
#define CHADCRb 1
#define BRANb 2
#define ADDCRb 3
#define DISCHRb 4
#define KTCb 5
#define STORb 6
#define BUFb 7




// кількість екранів при заряді на дисплей 1602
#if (DISPLAYy == 2)
#define WINDC 4
// кількість екранів на дисплей 2004
#elif (DISPLAYy == 4)
#define WINDC 2
#endif

#define InvBit(reg, bit) reg ^= (1 << bit) // nvBit(tmp,6); //інвертувати шостий біт змінної tmp
#define BitIsClear(reg, bit) ((reg & (1 << bit)) == 0)
//if (BitIsClear(PIND, 0)) { //якщо очищений нульовий біт у регістрі PIND //виконати блок



/*
Карта організації пам'яті EPRROM
```

```
0 - порожньо
1 - 23 - struct vkr 22 байти
30 - 105 - профіль 0: struct pam 75 байт
106 - 181 - профіль 1: struct pam 75 байт
181 - 256 - профіль 2: struct pam 75 байт
257 - 332 - профіль 3: struct pam 75 байт
333 - 408 - профіль 4: struct pam 75 байт
409 - 484 - профіль 5: struct pam 75 байт
485 - 560 - профіль 6: struct pam 75 байт
561 - 636 - профіль 7: struct pam 75 байт
637 - 712 - профіль 8: struct pam 75 байт
713 - 788 - профіль 9: struct pam 75 байт
789 – 848 – порожньо 59 байт
849 – поточна кількість циклів
850 - 866 - КТЦ-1, архів 16 байт int32_t КТС [4] = {0}; // Ah_discharge, Wh_discharge, Ah_charge, Wh_charge, КТЦ
867 - 883 - КТЦ-2
884 – 900 – КТЦ-3
901 - 917 - КТЦ-4
918 - 934 - КТЦ-5
935 - 951 - КТЦ-6
952 – 968 – КТЦ-7
967 – 985 – КТЦ-8
986 – 1002 – КТЦ-9
1003 – 1019 – КТЦ-10
*/
Налаштування  меню
// назви акумуляторів
const char name1[] PROGMEM = "Pb Ca/Ca";
const char name2[] PROGMEM = "Pb Ca+";
const char name3[] PROGMEM = "Pb Sur";
const char name4[] PROGMEM = "Pb AGM";
const char name5[] PROGMEM = "Pb Gel";
const char name6[] PROGMEM = "Li-ion";
const char name7[] PROGMEM = "LiFePo4";
const char name8[] PROGMEM = "LiTit";
const char name9[] PROGMEM = "NiCd/Mh";

// Оголошуємо таблицю посилань
const char* const typeAkb[] PROGMEM = {
 name1, name2, name3, name4, name5,
 name6, name7, name8, name9,
};

#define POINTMENU 9 // Число пунктів Меню
#define POINTMODE 7 // число пунктів Режимів роботи
#define POINTSETTINGS 20 // кількість пунктів Налаштувань
#define POINTCALIBRS 1 // число пунктів Калібрування
#define SYSPARAM 19 // номер системних параметрів
#if (INTERFACE == 0)
const char namm1[] PROGMEM = "Pr:"; // 0 Профіль
const char namm2 [] PROGMEM = "x45xBCxBAx6Fx63xBFxC4x41x4BxA0"; // 1 Ємність // АКБ
const char namm3[] PROGMEM = "x54xB8xBE \x41x4BxA0"; // 2 Тип АКБ
const char namm4[] PROGMEM = "x48x61xBEx70xC7xB6x65xBDxB8x65x41x4BxA0"; // 3 Напруга АКБ
const char namm5[] PROGMEM = "x50x65xB6xB8xBC x70x61xB2x6FxBFxC3"; // 4 Режим роботи
const char namm6[] PROGMEM = "x48x61x63xBFx70x6FxB9xBA8B"; // 5 Налаштування
const char namm7 [] PROGMEM = "x43x6FxBEx70x6FxBFx2Ex41x4BxA0"; // 6 Сопрот. АКБ
const char namm8[] PROGMEM = "x43xBFx61xBFxB8x63xBFxB8xBAx61"; // 7 Статистика
const char namm9[] PROGMEM = "x4Bx61xBBxB8xB2x70x6FxB3xBAx61"; // 8 Калібрування
const char namm10[] PROGMEM = " ";
const char* const menuTxt[] PROGMEM = {
 namm1, namm2, namm3, namm4, namm5,
 namm6, namm7, namm8, namm9, namm10,
};

const char namd1[] PROGMEM = "xA4x61x70xC7xE3"; // 0 Заряд
const char namd2[] PROGMEM = "A4x61x70x7xE3>xE0x6FxB7x61x70x7xE3"; // 1 Заряд>Дозаряд
const char namd3[] PROGMEM = "A4x61x70x7xE3xA0x70x61xBDxB8xBC8Bxx70x61"; // 2 Заряд Бранімір
const char namd4[] PROGMEM = "xE0x6FxB7x61x70xC7xE3"; // 3 Дозаряд
const char namd5[] PROGMEM = "x50x61xB7x70xC7xE3"; // 4 Розряд
```

```
const char namd6[] PROGMEM = "КТ\xE5"; // 5 КТЦ (Розр>Зар>Дозар)
const char namd7[] PROGMEM = "x58x70x61xBDx65xBDxB8x65"; // 6 Зберігання АКБ
const char namd8[] PROGMEM = "A0x79xE4.x70x65xB6xB8xBC"; // 7 Буф.режим // Буферний режим "A0x79xE4x65x70xBDxx3xB9x70x65xB6xB8xBC";

const char* const modeTxt[] PROGMEM = {
 namd1, namd2, namd3, namd4,
 namd5, namd6, namd7, namd8,
};

#if (DISPLAYy == 2)
const char regimr[] PROGMEM = "\xA4\xA4\xA0\xE0\x50K"; // Символи виведення режиму роботи - Заряд, Заряд, Бранімір, Дозаряд, Розряд, КТЦ
#endif
const char nami1[] PROGMEM = "\x48\x61\xBE\x70\xC7\xB6 \xB7\x61\x70\xC7\xE3\x61";            // 0  Напряж. заряда  *
const char nami2[] PROGMEM = "\x54\x6F\xBA \xB7\x61\x70\xC7\xE3\x61";                       // 1  Ток заряда      *
const char nami3[] PROGMEM = "\x48\x61\xBE\x70\xC7\xB6 \x43\xBD\xB8\xB6.";                  // 2  Напряжение при котором начинает снижаться ток
const char nami4[] PROGMEM = "\x4D\xB8\xBD. \xBF\x6F\xBA \xB7\x61\x70\xC7\xE3\x61";         // 3  Мин. ток заряда *
const char nami5[] PROGMEM = "\x42\x70\x65\xBC\xC7 \xB7\x61\x70\xC7\xE3\x61";               // 4  Время заряда    *
const char nami6[] PROGMEM = "\x48\x61\xBE\x70. \xE3\x6F\xB7\x61\x70\xC7\xE3\x61";          // 5  Напр. дозаряда  *
const char nami7[] PROGMEM = "\x54\x6F\xBA \xE3\x6F\xB7\x61\x70\xC7\xE3\x61";               // 6  Ток дозаряда    *
const char nami8[] PROGMEM = "\x4D\xB8\xBD. \xBF\x6F\xBA \xE3\x6F\xB7\x61\x70\xC7\xE3\x61"; // 7  Мин. ток дозар. *
const char nami9[] PROGMEM = "\x42\x70\x65\xBC\xC7 \xE3\x6F\xB7\x61\x70\xC7\xE3\x61";       // 8  Время дозаряда  *
const char nami10[] PROGMEM = "\x48\x61\xBE\x70. \x70\x61\xB7\x70\xC7\xE3\x61";             // 9  Наряж. разряда  *
const char nami11[] PROGMEM = "\x54\x6F\xBA \x70\x61\xB7\x70\xC7\xE3\x61";                  // 10 Ток разряда     *
const char nami12[] PROGMEM = "\x4D\xB8\xBD. \xBF\x6F\xBA \x70\x61\xB7\x70\xC7\xE3\x61";    // 11 Мин. ток разряда*
const char nami13[] PROGMEM = "\xA0\x79\xE4\x65\x70\xBD\xc3\xB9 \x70\x65\xB6\xB8\xBC";      // 12 Буферный режим  *
const char nami14[] PROGMEM = "\x58\x70\x61\xBD\x65\xBD\xB8\x65";                           // 13 Хранение
const char nami15[] PROGMEM = "\xA8\x70\x65\xE3\xB7\x61\x70\xC7\xE3";                       // 14 Предзаряд       *
const char nami16[] PROGMEM = "\x48\x61\xBE\x70. \xBE\x70\x65\xE3\xB7\x61\x70\xC7\xE3\x61"; // 15 Напряж. предзар.*
const char nami17[] PROGMEM = "\x54\x6F\xBA \xBE\x70\x65\xE3\xB7\x61\x70\xC7\xE3\x61";      // 16 Ток предзаряда  *
const char nami18[] PROGMEM = "\x4B\x61\xC0\x65\xBB\xB8";                                   // 17 Качели          *
const char nami19[] PROGMEM = "\xE1\xB8\xBA\xBB";                                           // 18 Цикл
const char nami20[] PROGMEM = "\x21\x43\xB8\x63\xBF\x65\xBC\x2E\x20\xBE\x61\x70\x61\xBC\x21"; // 19 !Систем. парам! *
const char nami21[] PROGMEM = "\x43\xB2\x70\x6F\x63 \xBD\x61\x63\xBF\x70\x6F\x65\xBA";      // 20 Сброс настроек  *

const char* const settings[] PROGMEM = {
  nami1, nami2, nami3, nami4, nami5, nami6, nami7, nami8, nami9, nami10, nami11,
  nami12, nami13, nami14, nami15,  nami16, nami17, nami18, nami19, nami20, nami21,
};

const char namc1[] PROGMEM = "\x48\x61\xBE\x70\xC7\xB6/\x54\x6F\xBA \x41\xBA\xB2";  // Напряж/ток акб
const char namc2[] PROGMEM = "\x48\x61\xBE\x70\xC7\xB6 \xA0\xA8";                   // Напряж. БП

const char* const calibr[] PROGMEM = { namc1, namc2 };

#define txt2 " \x63\xB2\x70\x6F\x63"                    //  сброс
#define txt3 " \x63\x6F\x78\x70\x21"                    //  сохр!
#define txt4 "\x42\x78\x2E\x20\xA0\xA8 "                //  "Вх. БП "
#define txt5 "\x42\xC3\x78\x6F\xE3"                     //  Выход
#define txt6 "INA226 \x6F\xC1\xB8\xB2\xBA\x61"          //  INA226
#define txt7 "DS18B20 "                                 //  DS18B20
#define txt8 "!ALERT!"                                  //  Темп -
//#define txt9 " cek"                                   //  сек
#define txt10 "\x42\x63\xA2"                            //  Всё
#define txt11 "\xA8\x6F\xE3\xBA\xBB\xC6\xC0\xB8 \x41\x4B\xA0"  //  Подключи АКБ
#define txt12 "\x43\xB2\x70\x6F\x63"                    //  Сброс
#define txt13 "\x42\xBA\xBB\x2E "                       //  Вкл.
#define txt14 "\x4F\xBF\xBA\xBB\x2E"                    //  Откл.
//#define txt15 "\xAB\x61\x63"                          //  Час
#define txt16 "\xA8\x61\x79\xB7\x61"                    //  Пауза
#define txt17 "\x3C\x4F\x4B\x3E"                        //  "< OK >"
//#define txt18 "V Bpe\xBC\xC7 "                        //  V Время
#define txt19 "\x3C\x43\xBF\x6F\xBE\x3e"                //  "<Стоп>"
#define txt20 "\xBC\xB8\xBD"                            //  мин
#define txt21 "\x6F\xC1\xB8\xB2\xBA\x61"                //  ошибка
#define txt22 "Volt"                                    //  Вольт
```

```
#define txt23 "Amper"                                          // Ампер
#define txt24 "!-\xA8\x50\x4F\xA0\xA5\x54 Q1-!"                //  !-ПРОБИТ Q1-!
#define txt25 "\x56\xB2\xBE "                                  //  Vбп
//#define txt26 "\x41\x4B\xA0\x20\x70\x61\xB7\x70\xC7\xB6\x65\xBD\x21"  //  АКБ разряжен!
//const char err[] PROGMEM = "\x6F\xC1\xB8\xB2\xBA\x61";

// английский
#elif (INTERFACE == 1)
const char namm1[] PROGMEM = "Pr:";
const char namm2[] PROGMEM = "Capacity Akb";
const char namm3[] PROGMEM = "Type Akb";
const char namm4[] PROGMEM = "Voltage Akb";
const char namm5[] PROGMEM = "Operating mode";
const char namm6[] PROGMEM = "Settings";
const char namm7[] PROGMEM = "Resistance Akb";
const char namm8[] PROGMEM = "Statistics";
const char namm9[] PROGMEM = "Calibration";
const char namm10[] PROGMEM = " ";

const char* const menuTxt[] PROGMEM = {
  namm1, namm2, namm3, namm4, namm5,
  namm6, namm7, namm8, namm9, namm10,
};

const char namd1[] PROGMEM = "Charge";
const char namd2[] PROGMEM = "Charge>AddChar";
const char namd3[] PROGMEM = "Charge Branim";
const char namd4[] PROGMEM = "Add charge";
const char namd5[] PROGMEM = "Discharge";
const char namd6[] PROGMEM = "Contr.trai.cycle";
const char namd7[] PROGMEM = "Storage";
const char namd8[] PROGMEM = "Buffer mode";  // "Buffer mode"

const char* const modeTxt[] PROGMEM = {
  namd1, namd2, namd3, namd4,
  namd5, namd6, namd7, namd8,
};

#if (DISPLAYy == 2)
const char regimr[] PROGMEM = "CCBADK";  // символы вывода режима работы - Заряд, Заряд, Бранимир, Дозаряд, Разряд,
КТЦ
#endif
const char nami1[] PROGMEM = "Volt charge"; // 0 Напруж. заряда
const char nami2[] PROGMEM = "Curr. charge"; // 1 Струм заряду
const char nami3[] PROGMEM = "Volt decr.Cur"; // 2 Напруга при якому починає знижуватися струм
const char nami4[] PROGMEM = "Curr. charge off"; // 3 струм завершення заряду
const char nami5 [] PROGMEM = "Time charge"; // 4 Час заряду
const char nami6[] PROGMEM = "Volt add charge"; // 5 Напруж. дозаряду
const char nami7[] PROGMEM = "Curr. add charge"; // 6 Струм дозаряду
const char nami8[] PROGMEM = "Curr.addChar min"; // 7 хв. Струм дозар.
const char nami9[] PROGMEM = "Time add charge"; // 8 Час дозаряду
const char nami10[] PROGMEM = "Volt dischar"; // 9 Убрання. розряду
const char nami11[] PROGMEM = "Curr. dischar"; // 10 Струм розряду
const char nami12[] PROGMEM = "Curr.disch. off"; // 11 Мін. струм розряду
const char nami13[] PROGMEM = "Bufer mode"; // 12 Буферний режим
const char nami14[] PROGMEM = "Storage mode"; // 13 напруга режиму Зберігання
const char nami15[] PROGMEM = "Precharge"; // 14 Передзаряд
const char nami16[] PROGMEM = "Volt pre charge"; // 15 Напруж. предзар.
const char nami17[] PROGMEM = "Curr. pre charge"; // 16 Струм передзаряду
const char nami18[] PROGMEM = "Oscillation"; // 17 Гойдалка
const char nami19[] PROGMEM = "Cycle"; // 18 Цикл
const char nami20[] PROGMEM = "!system param!"; // 19 #define SYSPARAM 18 // номер системних параметрів
const char nami21[] PROGMEM = "Reset settings"; // 20 Скидання налаштувань

const char* const settings[] PROGMEM = {
  nami1, nami2, nami3, nami4, nami5, nami6, nami7, nami8, nami9, nami10, nami11,
  nami12, nami13, nami14, nami15,  nami16, nami17, nami18, nami19, nami20, nami21,
};
```

```
const char namc1[] PROGMEM = "Volt/Curr akb";
const char namc2[] PROGMEM = "Volt Power";

const char* const calibr[] PROGMEM = {
  namc1,
  namc2,
};

#define txt2 " reset"
#define txt3 " saved"  // 2 раза
#define txt4 "Power "
#define txt5 "Exit"
#define txt6 "INA226 error"
#define txt7 "DS18B20 "
#define txt8 "!ALERT!"
//#define txt9 " sec"
#define txt10 "All"
#define txt11 "Connect Batt"
#define txt12 "Sbros"
#define txt13 "On "
#define txt14 "Off "
//#define txt15 " hour" // 2 раза
#define txt16 "Pause"
#define txt17 "< OK >"
//#define txt18 "V  Time "
#define txt19 "<Stop>"
#define txt20 " min "
#define txt21 "error"  // 6 раз
#define txt22 "Volt"
#define txt23 "Amper"
#define txt24 "!-BROKEN Q1-!"
#define txt25 "Vin "  //  Напряж БП превыш
//#define txt26 "Akb razryazen!"
//const char err[] PROGMEM = "error";

// транстит
#elif (INTERFACE == 2)
const char namm1[] PROGMEM = "Pr:";
const char namm2[] PROGMEM = "Emkost akb";
const char namm3[] PROGMEM = "Tip Akb";
const char namm4[] PROGMEM = "Napryagenie akb";
const char namm5[] PROGMEM = "Regim raboti";
const char namm6[] PROGMEM = "Nastroiki";
const char namm7[] PROGMEM = "Soprotivl. akb";
const char namm8[] PROGMEM = "Statistika";
const char namm9[] PROGMEM = "Kalibrovka";
const char namm10[] PROGMEM = " ";

const char* const menuTxt[] PROGMEM = {
  namm1, namm2, namm3, namm4, namm5,
  namm6, namm7, namm8, namm9, namm10,
};

const char namd1[] PROGMEM = "Zaryad";
const char namd2[] PROGMEM = "Zar.>Dozaryad";
const char namd3[] PROGMEM = "Zar. Branimir";
const char namd4[] PROGMEM = "Dozaryad";
const char namd5[] PROGMEM = "Razryad";
const char namd6[] PROGMEM = "KTC";
const char namd7[] PROGMEM = "Hranenie";
const char namd8[] PROGMEM = "Buf. regim";  // "Bufer regim"

const char* const modeTxt[] PROGMEM = {
  namd1, namd2, namd3, namd4,
  namd5, namd6, namd7, namd8,
};

#if (DISPLAYy == 2)
```

```
const char regimr[] PROGMEM = "ZZBDRK";  // символы вывода режима работы - Заряд, Заряд, Бранимир, Дозаряд, Разряд, КТЦ
#endif

const char nami1[] PROGMEM = "Napr. zaryada";
const char nami2[] PROGMEM = "Tok zaryada";
const char nami3 [] PROGMEM = "Napr. snigen"; // 2 Напруга при якому починає знижуватися струм
const char nami4[] PROGMEM = "Min. tok zaryda";
const char nami5 [] PROGMEM = "Vremya zaryada";
const char nami6[] PROGMEM = "Napr. dozaryada";
const char nami7[] PROGMEM = "Tok dozaryada";
const char nami8[] PROGMEM = "Min. tok dozar.";
const char nami9[] PROGMEM = "Vremya dozaryada";
const char nami10[] PROGMEM = "Napr. razryada";
const char nami11[] PROGMEM = "Tok razryada";
const char nami12[] PROGMEM = "Min. tok razr.";
const char nami13[] PROGMEM = "Bufernii regim";
const char nami14[] PROGMEM = "Xranenie Akb";
const char nami15[] PROGMEM = "Predzaryad";
const char nami16[] PROGMEM = "Напр. predzaryda";
const char nami17[] PROGMEM = "Ток predzaryda";
const char nami18 [] PROGMEM = "Regim kacheli";
const char nami19[] PROGMEM = "Cycle";
const char nami20[] PROGMEM = "!system param!";
const char nami21[] PROGMEM = "Sbros nastroek";

const char* const settings[] PROGMEM = {
 nami1, nami2, nami3, nami4, nami5, nami6, nami7, nami8, nami9, nami10, nami11,
 nami12, nami13, nami14, nami15, nami16, nami17, nami18, nami19, nami20, nami21,
};

const char namc1[] PROGMEM = "Volt/Curr akb";
const char namc2[] PROGMEM = "Volt Power";

const char * const calibr [] PROGMEM = { namc1, namc2 };

#define txt2 "sbros"
#define txt3 " sohr."
#define txt4 "Power"
#define txt5 "Exit"
#define txt6 "INA226 error"
#define txt7 "DS18B20"
#define txt8 "!ALERT!"
//#define txt9 " cek"
#define txt10 "Vse"
#define txt11 "Podkluchi Akb"
#define txt12 "Sbros"
#define txt13 "Vkl."
#define txt14 "Otkl."
//#define txt15 "chas"
#define txt16 "Pause"
#define txt17 "< OK >"
//#define txt18 "V Time"
#define txt19 "<Stop>"
#define txt20 "min"
#define txt21 "error"
#define txt22 "Volt"
#define txt23 "Amper"
#define txt24 "!-PROBIT Q1-!"
#define txt25 "Vin" // Напруж БП перевищив
//#define txt26 "Akb razryazen!"
//const char err[] PROGMEM = "error";
#endif
// ------ Сервісні параметри-----------
#define SETTINGS_AMOUNT 15 // кількість налаштувань
const char service0[] PROGMEM = "POWER";
const char service1[] PROGMEM = "POWER_MAX";
const char service2[] PROGMEM = "INAVKOOF";
const char service3[] PROGMEM = "CURR_MAX";
```

```
const char service4[] PROGMEM = "FAN";
const char service5[] PROGMEM = "DEG_ON";
const char service6[] PROGMEM = "DEG_OFF";
const char service7[] PROGMEM = "DEG_MAX";
const char service8[] PROGMEM = "DEG_CUR";
const char service9[] PROGMEM = "CURFAN_ON";
const char service10[] PROGMEM = "CURFAN_OFF";
const char service11[] PROGMEM = "CURFAN_MAX";
const char service12[] PROGMEM = "TIME_LIGHT";
const char service13[] PROGMEM = "FREQ_CHARGE";
const char service14[] PROGMEM = "FREQ_DISCHAR";


#define POWERINT ((uint16_t)vkr.service[0] * 1000) // Напруга від БП
#define POWERMAX 1 // максимальна напруга від БП
#define INAVKOOFV ((float)vkr.service[2] / 1000 + 1.0f) // коефіцієнт напруги INA226
#define CURRMAXINT ((int16_t)vkr.service[3] * 100) // максимальний струм
#define FANMODE 4 // режим роботи кулера
#define DEGON 5 // температура включення вентилятора у градусах
#define DEGOFF 6 // температура відключення вентилятора у градусах
#define DEGMAX 7 // значення температури для максимальних обертів вентилятора ШІМ.
#define DEGCUR 8 // температура при якій зменшується струм заряду
#define CURFANON 9 // значення струму в Амперах, при якому включається вентилятор. (До 25,5А)
#define CURFANOFF 10 // значення струму в Амперах, при якому відключається вентилятор. (До 25,5А)
#define CURFANMAX 11 // значення струму в Амперах для максимальних обертів вентилятора ШІМ. (До 25,5А)
#define TIMELIGHT 12 // час підсвічування дисплея у хвилинах
#define FREQCHARGE 13 // частота роботи силового модуля
#define FREQDISCHAR 14 // Частота роботи розрядного модуля

const char* const servicc[] PROGMEM = {
 service0, service1, service2, service3, service4, service5, service6, service7,
 service8, service9, service10, service11, service12, service13, service14,
};

// символи акумулятора та заряду
const char simb_array[8][8] PROGMEM = {
 {0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1F}, // akb 0
 {0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1F, 0x1F}, // akb 1
 {0x0E, 0x11, 0x11, 0x11, 0x11, 0x1F, 0x1F, 0x1F}, // akb 2
 {0x0E, 0x11, 0x11, 0x11, 0x1F, 0x1F, 0x1F, 0x1F}, // akb 3
 {0x0E, 0x11, 0x11, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F}, // akb 4
 {0x0E, 0x11, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F}, // akb 5
 {0x0E, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F}, // akb 6
 {0x03, 0x06, 0x0C, 0x18, 0x1F, 0x06, 0x0C, 0x18}, // заряд 7
};

//const char simbols[] PROGMEM = "VAWCNh%()<>:; ";



#define PROBEL 32 // пробіл
#define LEFTs 60 // <
#define RIGHTs 62 // >

#define txt_C "C"
#define txt_PRC "%"
#define txt_OK "OK"
#define txt_ViktoRi "ViktoRi"



Програма Меню дисплея LCD-1602
void Menu1602() {
 if (bitRead(pam.MyFlag, CHARGE)) return; // якщо прапор заряду встановлено true, то вийти інакше виконувати Menu
 Light1(true);
 pam.Number = 1;
 Saved(); // зберегти налаштування
 uint8_t urovenmenu = 0; // три рівні меню
 uint8_t main_menu = 0; // Номер пункту меню
```

```
 uint8_t point_setting = 0; // Номер пункту налаштувань
 bool printx = true;
 int8_t x = 0;
 int8_t tiks = 0;
 disp.start();
 ina.start(1); // старт вимірів INA
#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
#endif
 do {
 // цикл меню
 switch (urovenmenu) {
 // основне меню urovenmenu
 case 0:
 point_setting = 0;
 // Виведення меню
 if (printx) {
 printx = false;
 //if (++mode >= MODES) mode = 0; // на збільшення
 //if (--mode < 0) mode = MODES - 1; // на зменшення
 if (x) main_menu = (x > 0)? ((main_menu == POINTMENU) ? 0 : main_menu + x) : ((main_menu == 0) ? POINTMENU : main_menu
+ x); // Реалізація кругового меню
 lcd.clear();
 printFromPGM((int)(&menuTxt[main_menu]));
 setCursory();
 switch (main_menu) {
 case 0:
 // профіль
 print_Capacity();
 Vout(); // *Pr:0 Pb CA/Ca *
 lcd.print(vkr.profil); // *60Ah 12.60V(6) *
 lcd.setCursor(5, 0);
 print_mode(1); // Висновок типу акб
 break;
 case 1:
 // Ємність
 print_Capacity();
 break;
 case 2:
 // Тип акумулятора
 print_mode(1); // Висновок типу акб
 break;
 case 3:
 // напруга
 Vout();
 break;
 case 4:
 // Режим
 print_mode(0);
 break;
 }
 }
 if (main_menu <POINTMENU) {
 if (tiks == ENCCLICK or tiks == OKCLICK) {
 urovenmenu = (main_menu < 5)? 2: 1;
 point_setting = 0;
 printx = true;
 break;
 }
 // утримання Пуск або Енкодера запускає заряд

 if (tiks == ENCHELD or tiks == OKHELD) {
 // Бранімір, Дозаряд, КТЦ
 if (pam.Mode == ADDCRb and pam.typeAkb > 4) {
 lcd.clear();
 print_mode(1);
 printSimb(); // пробіл
 print_mode(0);
 setCursory();
```

```
print_mode(2); // "error"
Speaker(1000); // функція звукового сигналу (час у мілісекундах)
Delay (3000);
} else {
bitSet(pam.MyFlag, CHARGE); // запустити роботу режиму, вийти із циклу меню
}
printx = true;
}
}
break; // основне меню
case 1:
// urovenmenu 1 - підменю
switch (main_menu) {
case 0:
Saved(); // зберегти налаштування
urovenmenu = 0;
printx = true;
break;
case 1:
// ємність акб
case 2:
// тип акб
// Розрахунок при виборі ємності, типу та кількості осередків (напруги) Акб
// Розрахунок струмів
pam.Current_charge = constrain(bat.Cur(0), CUR_CHARGE_MIN, CURRMAXINT); // Струм заряду
pam.Current_charge_min = constrain((int16_t)pam.Capacity << 1, CURMIN, 200); // Струм завершення заряду
pam.Current_discharge = constrain(bat.Cur(2), 50, CURRMAXINT); // Струм розряду
pam.Current_discharge_min = pam.Current_discharge; // Струм завершення розряду
pam.Current_add_charge = constrain(bat.Cur(1), CUR_CHARGE_MIN, CURRMAXINT); // Струм дозаряду
pam.Current_add_charge_min = pam.Current_add_charge/3; // Мінімальний струм дозаряду
pam.Current_pre_charge = constrain(pam.Current_charge / 3, 200, CURRMAXINT); // Струм передзаряду
ChargeTimeCalc(); // Розрахунок часу заряду
// Напруга акб
//__attribute__((fallthrough));
case 3:
// Розрахунок напруг
{
pam.Volt_charge = bat.Volt(1); // напруга заряду, напруга заряду комірки кількість комірок
pam.Volt_decrCur = Volt_DCur(); // напруга у якому починається зниження струму заряду
pam.Volt_discharge = bat.Volt(3); // напруга розряду
uint16_t vbc = bat.Volt(0);
pam.Volt_storage = (uint16_t) (vbc * 1.04f); // напруга зберігання
pam.Volt_buffer = (uint16_t) (vbc * 1.1f); // напруга буферного режиму
if (pam.typeAkb < 5) { // розрахунок для свинцево кислотних Акб
pam.Volt_add_charge = bat.Volt(2); // напруга дозаряду
// Розрахунок струмів
pam.Current_charge = constrain((int16_t)pam.Capacity << 4, 150, 3500); // Струм заряду
pam.Current_charge_min = constrain((int16_t)(1.414f * pam.Capacity), CURMIN, 250); // Струм завершення заряду
//pam.Current_discharge = constrain(bat.Cur(2), 50, CURRMAXINT); // Струм розряду
//pam.Current_discharge_min = pam.Current_discharge; // Струм завершення розряду
pam.Current_add_charge = constrain(bat.Cur(1), CUR_CHARGE_MIN, CURRMAXINT); // Струм дозаряду
pam.Current_add_charge_min = pam.Current_add_charge/3; // Мінімальний струм дозаряду
//pam.Current_pre_charge = constrain(pam.Current_charge / 3, 200, CURRMAXINT); // Струм передзаряду
ChargeTimeCalc(); // Розрахунок часу заряду
if (pam.Mode == 2) pam.Round = 3; // для Браніміра мінімум 3 раунди
urovenmenu = 0;
printx = true;
break;
case 5:
// Налаштування
if (printx) {
printx = false;
if (x) point_setting = (x > 0)? ((point_setting == POINTSETTINGS) ? 0 : point_setting + x) : ((point_setting == 0) ? POINTSETTINGS
: point_setting + x);
lcd.clear();
printFromPGM((int)(&settings[point_setting]));
setCursory();
Settings_point(point_setting, false, 0); // Висновок/зміна пунктів налаштувань
}
```

```
 switch (tiks) {
 case STOPCLICK:
 case ENCHELD:
 urovenmenu = 0;
 point_setting = 0;
 printx = true;
 Saved(); // зберегти налаштування
 lcd.clear();
 print_mode(3); // "Settings"
 lcd.print(F(txt3)); // saved
 Delay (1000);
 break;
 case ENCCLICK:
 case OKCLICK:
 urovenmenu = 2;
 printx = true;
 break;
 }
 break;
 case 6:
 // Вимір опору
#if (RESIST == 1)
 Resist();
 Saved(); // зберегти налаштування
#endif
 urovenmenu = 0;
 printx = true;
 break;
 case 7:
 // статистика
 if (printx) {
 printx = false;
 if (x) point_setting = constrain(point_setting + x, 0, 3 + pam.Round);
 lcd.clear();
 Statistics_point(point_setting); // Висновок статистики
 }
 switch (tiks) {
 case ENCCLICK:
 case OKCLICK:
 case STOPCLICK:
 urovenmenu = 0;
 point_setting = 0;
 printx = true;
 break;
 }
 break;
 case 8:
 // калібрування
 if (printx) {
 printx = false;
 lcd.clear();
 lcd.setCursor(1, 0);
 printFromPGM((int)(&calibr[0]));
 lcd.setCursor(1, 1);
 printFromPGM((int)(&calibr[1]));
 if(x) point_setting = constrain(point_setting + x, 0, POINTCALIBRS);
 lcd.setCursor(0, point_setting);
 lcd.write(RIGHTs); // >
 }
 switch (tiks) {
 case STOPCLICK:
 case ENCHELD:
 urovenmenu = 0;
 point_setting = 0;
 printx = true;
 break;
 case ENCCLICK:
 case OKCLICK:
 urovenmenu = 2;
```

```
printx = true;
break;
}
break;
}
break;
case 2:
//urovenmenu – зміна параметрів
if (printx) {
printx = false;
setCursory();
if (main_menu) lcd.write(LEFTs); // <
switch (main_menu) {
case 0:
// Вибір профілю
vkr.profil = constrain(vkr.profil + x, 0, PROFIL);
EEPROM.get((int)(sizeof(pam) * vkr.profil + MEM), pam); // читаю з пам'яті значення
print_Capacity();
Vout();
lcd.write(LEFTs); // < // *Pr:<0>Pb CA/Ca *
lcd.print(vkr.profil);
lcd.write(RIGHTs); //>
print_mode(1); // Висновок типу акб
ClearStr(4); // Очистити поле
break;
case 1:
// Ємність
pam.Capacity = constrain(pam.Capacity + x, 1, 255);
lcd.print(pam.Capacity);
ClearStr(4); // Очистити поле
break;
case 2:
// Вибір типу акумулятора
pam.typeAkb = constrain(pam.typeAkb + x, 0, 8);
print_mode(1); // Висновок типу акб
ClearStr(4); // Очистити поле
break;
case
// Налаштування
Settings_point(point_setting, true, x); // Висновок/зміна пунктів налаштувань
if (point_setting == POINTSETTINGS) { // скидання налаштувань за умовчанням
urovenmenu = 0;
main_menu = 0;
printx = true;
}
if (tiks == OKCLICK or tiks == ENCCLICK or tiks == STOPCLICK or point_setting == SYSPARAM) {
urovenmenu = 1;
printx = true;
}
break;
case 8:
// калібрування
Korrect(point_setting);
Saved(); // зберегти налаштування
urovenmenu = 1;
printx = true;
break;
}
if (main_menu and main_menu < POINTSETTINGS - 2) {
lcd.setCursor(DISPLAYx - 1, 1);
lcd.write(RIGHTs); //>
}
}

if (main_menu < 5 and (tiks == ENCCLICK or tiks == OKCLICK or tiks == STOPCLICK)) {
urovenmenu = 1; //(main_menu == 4)? 0: 1;
printx = true;
}
break; // Зміна параметрів urovenmenu = 2
```

```
 }
 // - меню
 x = 0;
 // очікування дій користувача
 do {
 tiks = myTick();
 if (ina.sample()) {
#if (VOLTIN == 1)
 vin.sample(); // Вимірювання напруги БП
#endif
 }
 if (disp.tick()) {
 // якщо пройшла секунда виведення на дисплей
 Light1(false); // відключення підсвічування через 3 хвилини

 // Висновок на дисплей напруги, струму, температури раз на одну секунду
 if (main_menu == POINTMENU) {
 // * 24.12V 25C 20C *
 // * 12.652V 0.061A *
#if (VOLTIN == 1) // Напруга від БП
 setCursorx();
 PrintVA(vin.volt(), 0, 0, 0, 2);
#endif
 lcd.setCursor(7, 0);
#if (FAN)
 print_tr(kul.fan()); // Читання температури / управління кулером;
#elif (SENSTEMP1)
 print_tr(read_tq1()); // Читання температури
#endif
#if (SENSTEMP2 == 2)
 print_tr(tr_bat.getTempInt()); // Читання температури з датчика 2 акб
#endif

 setCursory();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 3); // * 12.361V -0.253A *
 }

#if (POWPIN == 1)
 Relay(); // Функція управління реле підключення до мережі 220В
#endif
#if (VOLTIN == 1)
 VinGuard(); // функція захисту від перевищення вхідної напруги

#if (POWPIN == 2)
 PowerLight(); // перевіряє напругу від БП і якщо вона більш ніж на Акб включає пін 10 (за замовчуванням), для світлодіода
#endif
#endif
 }
 // --- виконати раз на секунду
 } while (!tiks and !printx);
 // --- очікування дій користувача
 Light1(true);
 switch (tiks) {
 case ENCLEFT: / / поворот вліво
 case MINUSCLICK // кнопка мінус
 case MINUSSTEP:
 x = -1;
 break;
 case ENCRIGHT: // Поворот праворуч
 case PLUSCLICK: // кнопка плюс
 case PLUSSTEP:
 x = 1;
 break;
 }
 printx = true;
 } while (BitIsClear(pam.MyFlag, CHARGE)); // цикл меню

 if (pam.Mode! = 4) Res_mem_char (); // скидання значень заряду
 if (pam.Mode == 4 або pam.Mode == 5) Res_mem_dischar(); // скидання значень розряду
```

```
 if (pam.Mode == 5) Res_ktc(); // Очистити пам'ять КТЦ
 regim_end = 0;
 if (pam.Round <1) pam.Round = 1;
 EEPROM.put((MEM_KTC - 1), pam.Round); // зберегти кількість циклів
}
```

Операції об’єкта

```
void Operations() {
 const uint8_t mode = pam.Mode; // зберегти режим роботи
 // якщо прапор заряду встановлено true, то виконувати роботу
 while (bitRead(pam.MyFlag, CHARGE)) {
 uint8_t roundi;
 EEPROM.get((MEM_KTC - 1), roundi); // рахувати з пам'яті
 lcd.clear();
 print_mode(0); //Виведення режиму
 setCursory();
 Saved(); // зберегти налаштування
 Speaker(1000);
 // Вибір режиму заряду
 switch (pam.Mode) {
 case 0:
 // Заряд
 ChargeAkb(pam.Volt_charge, pam.Current_charge, pam.Current_charge_min, (uint32_t)pam.Time_charge_h, false); // заряд, дозаряд відкл
 if (bitRead(pam.MyFlag, CHARGE)) {
 roundi--;
 if(!roundi) pam.Mode = 6; // якщо лічильник циклів дорівнює 0, то завершити заряд // інакше повторити заряд
 }
 break;
 case 1:
 // Заряд - Дозаряд
 switch (pam.Number) {
 case 1:
 // Заряд
 ChargeAkb(pam.Volt_charge, pam.Current_charge, pam.Current_charge_min, (uint32_t)pam.Time_charge_h, false); // заряд дозаряд відкл
 pam.Number++;
 break;
 case 2:
 // Дозаряд
 ChargeAkb(pam.Volt_add_charge, pam.Current_add_charge, pam.Current_add_charge_min, (uint32_t)pam.Time_add_charge_h, true); // Включити дозаряд
 if (bitRead(pam.MyFlag, CHARGE)) {
 roundi--;
 if(!roundi) pam.Mode = 6; // якщо лічильник циклів дорівнює 0, то завершити заряд
 else pam.Number = 1; // інакше повторити заряд – дозаряд
 }
 break;
 }
 break;
 case 2:
 // Заряд по Браніміру
#if (BRANIMIR == 1)
 switch (pam.Number) {
 case 1:
// Бранімір Заряд 20 годин після досягнення максимальної напруги
 ChargeAkb(pam.Volt_charge, pam.Current_charge, pam.Current_charge_min, (uint32_t)pam.Time_add_charge_h, false); // заряд
(Напр. заряду, струм заряду, струм завершення заряду, час заряду, дозар. вимк.

 if (bitRead(pam.MyFlag, BRANIM)) pam.Number = 3; // якщо струм заряду знизився до встановленого, то переходимо в дозаряд
 else if (bitRead(pam.MyFlag, BRANIMBOIL)) pam.Number++; // інакше якщо час заряду перевищив 20 годин
 if(!roundi) pam.Number = 3; //або якщо лічильник циклів дорівнює 0, то перейти в дозаряд
 break;
```

```
 case 2:
 bitClear(pam.MyFlag, BRANIMBOIL);
 // Бранімір перемішування
 ChargeAkb(pam.Volt_add_charge + 1000, constrain((int16_t)pam.Capacity * 20, 150, 5000), constrain(((int16_t)pam.Capacity <<
3), CURMIN, 1500), 1, // перемішування (Напр. заряду, струм заряду, струм закінчення заряду, час заряду, дозар. вимк.
 roundi--;
 break;
 case 3:
 // Бранімір дозаряд
 if (pam.typeAkb < 5) ChargeAkb(pam.Volt_add_charge, pam.Current_add_charge, CUR_CHARGE_MIN, 10, true); // дозаряд
(Напр. заряду, струм заряду, струм закінчення заряду, час заряду, дозар. вкл.
 pam.Mode = 6;
 break;
 }
#else
 bitClear(pam.MyFlag, CHARGE);
#endif
 break;
 case 3:
 // Дозаряд
 ChargeAkb(pam.Volt_add_charge, pam.Current_add_charge, pam.Current_add_charge_min, (uint32_t)pam.Time_add_charge_h,
true); // Включити дозаряд
 if (bitRead(pam.MyFlag, CHARGE)) {
 roundi--;
 if(!roundi) pam.Mode = 6; // якщо лічильник циклів дорівнює 0, то завершити дозаряд
 }
 break;
 case 4:
 // Розряд
#if (DISCHAR == 1)
 Discharge(); //Розряд акумулятора
 if (bitRead(pam.MyFlag, CHARGE)) {
 roundi--;
 if (!roundi) bitClear(pam.MyFlag, CHARGE); // якщо лічильник циклів дорівнює 0, то завершити розряд // інакше повторити
розряд
 }
#else
 bitClear(pam.MyFlag, CHARGE);
#endif
 break;
 case 5:
 // Котнтрольно-тренувальний цикл
#if (DISCHAR == 1)
 {
 int32_t KTC[4] = {0}; // Ah_discharge, Wh_discharge, Ah_charge, Wh_charge,
 switch (pam.Number) {
 case 1:
 // Попередній заряд
 ChargeAkb(pam.Volt_charge, pam.Current_charge, pam.Current_charge_min, (uint32_t)pam.Time_charge_h, false); // заряд, доза-
ряд відкл
 pam.Number++;
 break;
 case 2:
 // дати відстоятися
 Wait(((uint16_t)pam.Capacity << 1)); // Функція очікування – хвилин
 pam.Number++;
 break;
 case 3:
 // Розряд
 Res_mem_dischar(); // скидання значень розряду
 Discharge(); //Розряд акумулятора
 pam.Number++;
 KTC [0] = pam.Ah_discharge; // Ah_discharge, Wh_discharge, Ah_charge, Wh_charge,
 KTC[1] = pam.Wh_discharge;
 EEPROM.put((((uint16_t)roundi << 4) - 16 + MEM_KTC), KTC); // Зберегти у пам'яті значення розряду н-циклу КТЦ
 break;
 case 4:
 // заряд
 Res_mem_char(); // скидання значень заряду
```

```
 ChargeAkb(pam.Volt_charge, pam.Current_charge, pam.Current_charge_min, (uint32_t)pam.Time_charge_h, false); // заряд, доза-
ряд відкл
 KTC [2] = pam.Ah_charge; // Ah_discharge, Wh_discharge, Ah_charge, Wh_charge,
 KTC [3] = pam.Wh_charge;
 EEPROM.put((((uint16_t)roundi << 4) - 16 + MEM_KTC), KTC); // Зберегти у пам'яті значення розряду н-циклу КТЦ
 if (bitRead(pam.MyFlag, CHARGE)) {
 roundi--;
 if(!roundi) pam.Number++; // якщо лічильник циклів дорівнює 0, то перейти в Дозаряд
 else pam.Number = 2; // інакше повторити КТЦ
 }
 break;
 case 5:
 // Дозаряд
 if (pam.typeAkb < 5) ChargeAkb(pam.Volt_add_charge, pam.Current_add_charge, pam.Current_add_charge_min,
(uint32_t)pam.Time_add_charge_h, true); // Включити дозаряд
 pam.Mode = 6; // Перейти до Зберігання
 break;
 }
 }
#else
 bitClear(pam.MyFlag, CHARGE);
#endif
 break;
 case 6: // зберігання
 Storage(true);
 pam.Mode = mode;
 bitClear(pam.MyFlag, CHARGE);
 break;
 case 7: // Буферний режим
 Storage(false);
 pam.Mode = mode;
 bitClear(pam.MyFlag, C
```

```
ADCSred
// Цей клас необхідний для зчитування напруги від БП та обчислення середнього значення
#define SOOT (((float)DIV_R1 + DIV_R2) / DIV_R2 / BITRATE)
class ADCSred {
public:
 // Список членів, доступних у програмі

 // ініціалізація (номер аналогового піна)
 ADCSred(uint8_t pin): _pin(pin) {}; // Номер піна

 // кожні prd мкс зчитуємо напругу БП (референсна напруга)
 void begin(uint16_t j) {
 _vref = j;
 }

 // Запуск вимірів.
 void start(void) {
 _vinmcrs = (int32_t)analogRead_my(_pin);
 }

 // зчитує напругу і підсумовує ковзну середню. Викликається разом із вимірами INA226 (ina.sample())
 void sample(void) {
 _vinmcrs += ((((int32_t)analogRead_my(_pin) << 4) - _vinmcrs) >> 4); // ковзне середнє кожні _prd мкс
 }

 // Повертає реальну напругу
```

```
 uint16_t volt(void) {
 return (uint16_t)((_vinmcrs >> 4) * SOOT * _vref); // розрахувати середню напругу за vr мкс
 }

private:
 // Список членів для використання всередині класу
 const uint8_t _pin; // Номер піна
 uint16_t _vref; // Референсна напруга
 int32_t _vinmcrs; // Середнє напруження за vr мкс
};
```

Інформація по заряду

```
// Прапори керування зарядом
#define CHARGEZ 0 // заряд увімкнено
#define OSCILZ 1 // осциляція
#define TAKBZ 2 // температура акб
#define KORR 3 // коригування напруги та струму заряду
#define KORRVA 4 // зміна вольт чи ампер
#define POWOFF 5 // живлення увімк./вимкнути
#define TMPQ1 6 // перевищена температура Q1
#define VMAXM 7 // напруга досягла максимуму

#include "DCDC.h" // клас управління DC-DC модулем

struct VA_var {
 uint16_t vmax[13] = {0}; // 13 значення напруги. Зберігаються кожні 5 хвилин.
 int16_t amin[13] = {0}; // 13 значення струму

 void sred_v(void) {
 vmax[12] += (((int16_t)ina.voltsec - (int16_t)vmax[12]) >> 2);
 }
 void sred_a(void) {
 amin [12] + = ((ina.ampersec - amin [12]) >> 2);
 }

 void VA_sred(void) {
 sred_v();
 sred_a();
 }

 void offset (void) {
 // зберегти 12 значень напруги та струму за 1 годину
 // Зміщення значень на одне вліво
 for (uint8_t n = 1; n < 12; n++) {
 vmax [n] = vmax [n + 1];
 amin [n] = amin [n + 1];
 }
 }
#if (GRAFIK == 1)
 void grafik(void) {
 // 1 - знайти мін та макс
 int16_t vmin = vmax [1];
 int16_t vmks = vmax [1];
 int16_t amnm = amin[1];
 int16_t amks = amin[1];
 for (uint8_t n = 2; n <= 12; n++) {
 if (vmax[n] and (vmin > (int16_t)vmax[n])) vmin = (int16_t)vmax[n]; // мінім напр
 if (vmax[n] and (vmks <(int16_t)vmax[n])) vmks = (int16_t)vmax[n]; // максим напр
 if (amin[n] and (amnm > amin[n])) amnm = amin[n]; // Мінім струм
 if (amin[n] and (amks <amin[n])) amks = amin[n]; // Макс струм
 }
 // Знайти різницю
 int16_t vraznica = vmks - vmin;
 int16_t araznica = amks - amnm;
```

```
 // 2 - знайти дозвіл
 int16_t vrazr = (vraznica > 700)? vraznica: 700;
 int16_t arazr = (araznica > 700)? araznica : ((amks <= 200 and araznica <= 200) ? 400 : 700);

 // 3 - знайти мінімальне значення
 vmin -= ((vrazr - vraznica) >> 1);
 amnm -= ((arazr - araznica) >> 1);
 if (vmin <0) vmin = 0;
 if (amnm < 0) amnm = 0;
 // 4 - знайти максимальне значення
 vmks = (vmin)? vmks + ((vrazr - vraznica) >> 1): vrazr;
 amks = (amnm)? amks + ((arazr - araznica) >> 1): arazr;

 for (uint8_t n = 1; n <= 12; n++) { // виведення на дисплей
 if (vmax [n]) {
 lcd.setCursor(n - 1, 0);
 lcd.write(map(vmax[n], vmin, vmks, 0, 6));
 lcd.setCursor(n - 1, 1);
 lcd.write(map(amin[n], amnm, amks, 0, 6));
 }
 }
 lcd.setCursor(12, 0);
 lcd.print(F(txt22)); // Volt
 lcd.setCursor(12, 1);
 lcd.print(F(txt23)); // Amper
 }
#endif
};

// Функція заряду. volt_charge_init - напруга заряду, curr_charge_init - струм заряду, curr_min - мінімальний струм заряду, time_charge - час заряду в годинах, add_charge - дозаряд увімкн.
void ChargeAkb(uint16_t volt_charge_init, int16_t curr_charge_init, int16_t curr_min, uint32_t time_charge, bool add_charge) {
 //виведення режиму, напруги та струму заряду/розряду
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
 // Висновок значень заряду
 PrintVA(volt_charge_init, curr_charge_init, 0, 0, 1); // Висновок напруги та струму заряду
 lcd.print(add_charge ? pam.Time_add_charge_h : pam.Time_charge_h); // Виведення часу заряду/дозаряду
 lcd.write(104); // h
 lcd.setCursor(DISPLAYx - 2, 0);
 printCykl(); // Виведення кількості циклів

 Freq(vkr.service[FREQCHARGE]); // Встановити частоту роботи силового модуля
 time_charge * = 3600; // перевів годинник за секунди
 uint32_t time_local = add_charge? pam.Time_addcharge : pam.Time_charge;

 VA_var va;

 const uint8_t VoltinKooff = bat.Volt(1) >> 8; // 14700/256 = 57 // pam.Voltage * 15; // кількість банок * 15мВ; // 4 * 15 = 60
 uint8_t typez = add_charge? 3 : (bitRead(pam.MyFlag, PRECHAR) ? 1 : 2); // 1 - Передзаряд, 2 - Заряд, 3 - Дозаряд (4 - осцилляція (гойдалка)), 5 - заряд по Бранімір
#if (BRANIMIR == 1)
 if (pam.Mode == 2) typez = 5; // 5 - заряд по Бранімір
 uint32_t tyme_branim = 0;
#endif
 uint8_t ap[2] = {4, 15}; // Коефіцієнт струму для передзаряду
 // задає напругу та струм заряду
 dcdc.begin(volt_charge_init, (add_charge ? curr_charge_init : (bitRead(pam.MyFlag, PRECHAR) ? pam.Current_pre_charge : curr_charge_init)));

 int8_t tr_c = 0; // значення температури
 uint8_t n_disp = 0, ns_disp = 0; // Номер 'екрану, час відображення
 uint16_t Timezar = 0; // Час до завершення заряду секунди
 uint16_t v1 = 0; // Змінна для збереження напруги
 uint8_t tsw = 10; // Дозаряд змінна часу перемикання, с.
 uint8_t timeaut10 = 2; // час 10 хвилин
 uint16_t timeaut5 = 300; // час 5 хвилин
 uint8_t time_millis = 0;
```

```
 uint16_t volt_init;
#if (VOLTIN == 1)
 uint16_t vin_volt;
#endif
 const int16_t cur_max = curr_charge_init;

#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, HIGH); // увімкнути мережу 220В
#endif
#if (LOGGER > 0)
 uint8_t logg=LOGGTIME; // період відправлення даних у мережевий порт
#endif
 int8_t tiks = 0;
 const int x[4] = { (int)(&settings[14]), (int)(&modeTxt[0]), (int)(&modeTxt[3]), (int)(&settings[17]) }; // {Попередній заряд, заряд,
дозаряд, Гойдалка}
 uint8_t flagsz = 0b00110111;
 // #define CHARGEZ 0 // 1 заряд увімкнено
 // #define OSCILZ 1 // 1 осциляція
 // #define TAKBZ 2 // 1 температура акб
 // #define KORR 3 // 0 коригування напруги та струму заряду
 // #define KORRVA 4 // 1 зміна вольт або ампер
 // #define POWOFF 5 // 1 живлення вкл/вимк.
 // #define TMPQ1 6 // 0 перевищена температура Q1
 // #define VMAXM 7 // напруга досягла максимуму
 uint8_t q1 = 0; // кількість перевірок силового транзистора 2
 Delay (3000);
 lcd.clear();
#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
 uint8_t vin_power_off = 5; // кількість секунд до того як спрацює умова
#endif
 disp.start(); // старт відліку часу одна секунда
 ina.start(1); // старт вимірів INA
 dcdc.start();
 // цикл заряду
 while (bitRead(flagsz, CHARGEZ) and bitRead(pam.MyFlag, CHARGE)) {
 tiks = myTick(); // опитування кнопок та енкодера

 if (tiks) {
 // якщо кнопка натиснута
 if (bitRead(flagsz, KORR)) {
 // проводиться коригування напруги та струму заряду
 int8_t k = 0;
 switch (tiks) {
 case ENCRIGHT: // енкодер праворуч
 case PLUSCLICK:
 case PLUSSTEP:
 k = 100;
 break;
 case ENCLEFT: // енкодер вліво
 case MINUSCLICK:
 case MINUSSTEP:
 k = -100;
 break;
 case ENCCLICK: // клік енкодер
 case OKCLICK // натиснути Пуск
 InvBit(flagsz, KORRVA);
 break;
 case ENCHELD: // утримати енкодер
 case STOPCLICK // натиснути Стоп
 case OKHELD // утримати Пуск
 bitClear(flagsz, KORR); //korr = false; // вийти із зміни
 setCursory();
 ClearStr(15); // очистити другий рядок
 break;
 }
 if (k) {
 if (bitRead(flagsz, KORRVA)) volt_charge_init = constrain(volt_charge_init + k, 1000, POWERINT);
 else curr_charge_init = constrain (curr_charge_init + k, CUR_CHARGE_MIN, CURRMAXINT);
```

```
 dcdc.begin(volt_charge_init, curr_charge_init);
 }
 } else {
 In
if (disp.tick()) {
 time_millis = 13; // раз на секунду запускаємо 13 кроків
 volt_init = (ina.voltsec> = (volt_charge_init - VoltinKooff)) ? volt_charge_init : ina.voltsec; // стабілізація максимальної напруги
#if (VOLTIN == 1)
 vin_volt = vin.volt();
#endif
 }

 // якщо минула секунда
 if (time_millis) {
 switch (time_millis) {
 case 13:
#if (VOLTIN == 1)
 VinGuard(); // функція захисту від перевищення вхідної напруги
#endif
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
 break;
 // якщо харчування від БП надходить
 if (bitRead(flagsz, POWOFF)) {
 case 12:
 // додати секунду до часу, якщо час більше встановленого завершити роботу
 if ((time_local++) > time_charge) {
 bitClear(flagsz, CHARGEZ);
 regim_end = 1; // Закінчився час
 }
 if (add_charge) {
 pam.Ah_addcharge += aw_ch((int32_t)ina.ampersec); // Розрахунок ампер/годин
 pam.Wh_addcharge += aw_ch(ina.getPower()); // Розрахунок ват/годин
 pam.Time_addcharge = time_local;
 } else {
 pam.Ah_charge += aw_ch((int32_t)ina.ampersec); // Розрахунок ампер/годин
 pam.Wh_charge += aw_ch(ina.getPower()); // Розрахунок ват/годин
 pam.Time_charge = time_local;
 }

 // виконати якщо увімкнено таймер завершення заряду
 if (Timezar) {
 Timezar--;
 if (!Timezar) bitClear(flagsz, CHARGEZ); // якщо таймер вийшов, то завершити заряд
 }
 break;
 case 11:
 switch (typez) {
 // Вибір типу заряду
 case 1:
 // Увімкнено попередній заряд
 dcdc.setCur_charge(map(timeaut5, 300, 1, constrain((int16_t)pam.Capacity * ap[0], 100, curr_charge_init),
constrain((int16_t)pam.Capacity * ap[1], 300, 300) ); // Протягом 300сек збільшуємо струм заряду від ap[0]З до ap[1]С
 if (timeaut5 > 250 and (va.vmax[12] < volt_init)) va.VA_sred();
 break;
 case 2:
 // Увімкнено заряд
 va.VA_sred(); // ковзне середнє - коливання напруги та струму зменшено в 4 рази
 // почати зниження струму заряду при досягненні величини напруги половини максимального
 // починає працювати коли напруга досягне встановленого струму і заряду більше 1% від ємності
 if (va.vmax[12] > pam.Volt_decrCur and va.amin[12] > (int16_t)pam.Capacity * 10) {
 curr_charge_init = map(va.vmax[12], pam.Volt_decrCur, volt_charge_init, cur_max, (int16_t)pam.Capacity * 10);
 dcdc.begin(volt_charge_init, curr_charge_init);
 }
 break;
 case 3:
 // Дозаряд
 // якщо напруга стабілізувалася, час завершився
```

```
 if (abs((int16_t)volt_init - (int16_t)v1) < 10 and !tsw) {
 // якщо максимальний струм
 if (dcdc.getCur_charge() == curr_charge_init) {
 va.VA_sred(); // ковзне середнє - коливання напруги та струму зменшено в 4 рази
 dcdc.setCur_charge(pam.Current_discharge_min); // міняємо струм заряду curr_charge_init / 3
 } else dcdc.setCur_charge(curr_charge_init); // інакше змінюємо струм заряду
 tsw = 10; // Очікування 10 сек
 } else v1 = volt_init;
 if (tsw) tsw--;
 break;
 case 4:
 // якщо в налаштуваннях включена Осциляція (Гойдалка) то при зниженні струму менше curr_min * 2 включається Режим осциляції - включення та відключення струму
 if (abs((int16_t)volt_init - (int16_t)v1) < 10) {
 // якщо струм увімкнено
 if (bitRead(flagsz, OSCILZ)) {
 // ковзне середнє - коливання напруги та струму зменшено в 4 рази
 if (va.vmax[12] <volt_init) va.sred_v(); // зберігаємо максимальну напругу vmax[12] = volt_init
 if (va.amin[12] > ina.ampersec or !va.amin[12]) va.sred_a(); // зберігаємо мінімальний струм amin[12] = ina.ampersec
 }
 InvBit (flagsz, OSCILZ); // інвертуємо прапор
#if (MCP4725DAC)
 if (BitIsClear(flagsz, OSCILZ)) dac.setVoltage(0);
#else
 if (BitIsClear(flagsz, OSCILZ)) digitalWrite_speed(PWMCH, LOW);
#endif
 } else v1 = vol
switch (n_disp) {
 // Екран 0
 case 0:
 Display_print((add_charge ? pam.Ah_addcharge : pam.Ah_charge), time_local, Percentage(volt_charge_init, va.vmax[12], va.amin[12], curr_min), add_charge); // Висновок поточних значень напруги, струму. Висновок Ватт-годин.
 break;
 case 1:
 // Екран 1
 setCursorx();
 print_mode(0);
 setCursory();
 printFromPGM(x[typez - 1]);
 printSimb(); // пробіл
 lcd.print(Percentage(volt_charge_init, va.vmax[12], va.amin[12], curr_min)); // Відсоток заряду Акб
 lcd.print(F(txt_PRC));
 lcd.print(dcdc.getTok()); // Висновок значення ШІМ заряду
 break;
 // *Charge>AddChar *
 // * Charge 45% 125 *
 case 2:
 // Екран 2
 setCursorx();
 PrintVA(0, 0, 0, (add_charge ? pam.Wh_addcharge : pam.Wh_charge), 2);
 print_tr(tr_c); //температура
 lcd.setCursor(DISPLAYx - 1, 0);
 lcd.print(regim_end); // причина завершення заряду
#if (SENSTEMP2 == 2)
 print_tr(tr_bat.getTempInt()); //температура
#endif
 setCursory();
#if (VOLTIN == 1)
 PrintVA(vin_volt, 0, 0, 0, 1);
#endif
 printCykl();
 if (Timezar) {
 printSimb();
 lcd.print(Timezar); // виконати якщо увімкнено таймер завершення заряду
 }
 break;
 // * 58.32Wh 45C 25C *
 // * 24.1V C1 N1 *
 case 3:
```

```
 // Екран 3
#if (GRAFIK == 1)
 if (ns_disp == TIME_DISP) va.grafik();
#else
 setCursorx();
 lcd.print((float)(ina.getPower() / 1000), 2);
 lcd.print(F("W"));
#endif
 break;
 // Зміна напруги і струму заряду
 case 4:
 // Екран 4
 setCursorx();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 2); // поточна напруга та струм
 lcd.setCursor(DISPLAYx - 2, 0);
 lcd.print(F(txt_OK));
 if (bitRead(flagsz, KORR)) {
 setCursory();
 Write_x(bitRead(flagsz, KORRVA)); // > or пробіл
 PrintVA(volt_charge_init, 0, 0, 0, 1); // Напруга та струм заряду
 Write_x(!bitRead(flagsz, KORRVA)); // > or пробіл
 PrintVA(0, curr_charge_init, 0, 0, 1); // Напруга та струм заряду
 }
 break;
 default:
 n_disp = 0;
 lcd.clear(); // Очистити дисплей
 break;
 }
#endif
// Висновок на дисплей 2004
#if (DISPLAYy == 4)
 switch (n_disp) {
 // Екран 0
 case 0:
 Display_print((add_charge ? pam.Ah_addcharge : pam.Ah_charge), (add_charge ? pam.Wh_addcharge : pam.Wh_charge),
time_local, tr_c, Percentage(volt_charge_init, va.vmax[12], va.amax );
 break;
 case 1:
 setCursorx();
 print_mode(0);
 setCursory();
 printFromPGM(x[typez - 1]);
 lcd.setCursor(0, 2);
 lcd.print(regim_end); // причина завершення заряду
 if (Timezar) {
 printSimb();
 lcd.print(Timezar); // виконати якщо увімкнено таймер завершення заряду
 }
 printCykl();
 break;
 // Зміна напруги і струму заряду
 case 2:

setCursorx();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 2); // поточна напруга та струм
 lcd.setCursor(DISPLAYx - 2, 0);
 lcd.print(F(txt_OK));
 if (bitRead(flagsz, KORR)) {
 setCursory();
 Write_x(bitRead(flagsz, KORRVA));
 PrintVA(volt_charge_init, 0, 0, 0, 1); // Напруга та струм заряду
 Write_x(!bitRead(flagsz, KORRVA));
 PrintVA(0, curr_charge_init, 0, 0, 1); // Напруга та струм заряду
 }
 break;
#if (GRAFIK == 1)
 case 3:
 // Висновок графіка
```

```
 if (ns_disp == TIME_DISP) va.grafik();
 break;
#endif
 default:
 n_disp = 0;
 break;
 }
#endif
 if (BitIsClear(flagsz, KORR) and n_disp) {
 if (ns_disp) ns_disp--;
 else {
 n_disp = 0;
 lcd.clear();
 }
 }
 break;
 case 9:
 if (volt_init > va.vmax[0]) va.vmax[0] = volt_init; // зберегти в vmax[0] максимальну напругу
 if (ina.ampersec > va.amin[0]) va.amin[0] = ina.ampersec; // зберегти bmeter.voltn[0] максимальний струм
 // якщо прапор макс напр не встановлений і напруга досягла максимального, то встановити цей прапор
 if (BitIsClear(flagsz, VMAXM) та (va.vmax[0] >= volt_charge_init - 50)) bitSet(flagsz, VMAXM);
 // інакше якщо прапор макс напр встановлені і напруга стала нижчою за максимальну то зняти цей прапор
 else if (bitRead(flagsz, VMAXM) and (volt_init < volt_charge_init - 500)) {
 bitClear(flagsz, VMAXM);
 va.vmax[0] = 0;
 Timezar = 0;
 }
 break;
 case 8:
 timeaut5--;
 // Виконати кожні 5 хвилин
 if (!timeaut5) {
 timeaut5 = 300;
 // зберегти 12 значень напруги та струму за 1 годину
 va.offset(); // Зміщення значень на одне вліво

 switch (typez) {
 // Увімкнено попередній заряд
 case 1:
 ap[1] += 3; // кожні 5 хвилин збільшуємо струм заряду.
 ap[0] = ap[1]/3;
 // якщо напруга більша за напругу передзаряду то відключити попередній заряд
 if (va.vmax[12] > pam.Volt_pre_charge) {
 typez++; // Вимкнути попередній заряд - вкл. звичайний заряд
 dcdc.setCur_charge(curr_charge_init);
 }
 break;
 case 2:
 // Увімкнено заряд
 // увімкнення режиму Очиляція (Гойдалки) при зниженні струму заряду до curr_min * 3, не виконується заряд по Бранімір
 if (bitRead(pam.MyFlag, OSCIL) and (va.amin[12] < constrain(curr_min << 1, curr_min, (int16_t)pam.Capacity * 7))) typez = 4;
 //__attribute__ ((fallthrough));
 case 3:
 // включений дозаряд,
 //якщо напруга досягла максимального
 if (bitRead(flagsz, VMAXM)) {
 // Треба стежити за струмом
 // якщо він не зменшується, то включити таймер на 1 годину

 // якщо минув час
 uint8_t ag = 1;
 //Шукаємо заповнений осередок
 while (ag < 11 and va.amin[ag] == 0) ag++;
 // Чим ближче струм до мінімального, тим більше проміжок часу
 uint8_t an = map(va.amin[12], curr_charge_init, curr_min, 11, ag);
 uint8_t cm = map(va.amin[12], curr_charge_init, curr_min, 50, 10);

 if (va.amin[12] <= curr_min) {
 // якщо струм знизився менше струму відключення, то завершити заряд, дозаряд
```

```
 if (regim_end != 2 or !Timezar) Timezar = 600; // Включаємо таймер 10 хв
 // якщо струм продовжує зменшуватись
 if ((va.amin[11] - va.amin[12]) > cm) Timezar = 600; // продовжуємо час
 regim_end = 2; // напруга досягла максимуму струм мінімуму
 //bitClear(flagsz, CHARGEZ);
 } else {
 if (va.amin[an] - va.amin[12] < cm) {
 regim_end = 5; // напруга досягла максимуму, струм перестав зменшуватися
 if (! Timezar) Timezar = 3600;
 } else if (Timezar) Timezar = 3600;

 if ((va.amin[an] - va.amin[12]) <-cm) {
 regim_end = 4; // напруга досягла максимуму, струм почав збільшуватися
 if (! Timezar) Timezar = 3600;
 } else if (regim_end == 4 and Timezar) {
 Timezar = 0;
 regim_end = 0;
 }
 /*
 if (((int16_t)vmax[1] - (int16_t)vmax[12]) > (int16_t)(Vbatt(CellCharge[pam.typeAkb]) >> 7)) {
 regim_end = 3; // Напруга досягла максимуму і почало знижуватися;
 if (! Timezar) Timezar = 3600;
 } else if (Timezar) Timezar = 3600;
 */
 }
 }
 break;
 case 4:
 // Увімкнено режим Гойдалки
 // якщо мінімальний струм менше струму завершення заряду і напруга досягла максимального, то зменшити струм завершення заряду
 if ((va.amin[12] < curr_min) and bitRead(flagsz, VMAXM)) {
 // якщо струм зменшився на 30мА
 if (curr_min - va.amin [12] > 30) {
 curr_min = va.amin [12]; // зменшити струм завершення заряду
 Timezar = 1200; // і засікти час
 regim_end = 6;
 }
 // якщо час завершення заряду не включено, то засікти час
 if (! Timezar) Timezar = 1200; // і засікти час
 }
 break;
 case 5:
 // заряд по Браніміру
#if (BRANIMIR == 1)
 // якщо напруга досягла максимального починаємо відлік часу
 if (bitRead(flagsz, VMAXM)) {
 tyme_branim + = 300;
 if (tyme_branim >= 72000) {
 bitClear(flagsz, CHARGEZ); // якщо минуло більше 20 годин, зупиняємо заряд і перемішуємо
 bitSet(pam.MyFlag, BRANIMBOIL); // встановити прапор перемішування електроліту
 }
 if (va.amin[12] <= curr_min) {
 // якщо струм заряду знизився до встановленого, то встановлюємо прапор про успішний заряд Браніміра і переходимо в дозаряд.
 regim_end = 2; // напруга досягла максимуму струм мінімуму;
 bitSet(pam.MyFlag, BRANIM); // встановити прапор успішного завершення заряду по Бранімір
 bitClear(flagsz, CHARGEZ); // Вимкнути заряд
 }
 }
#endif
 break;
 }
 timeaut10--;
 // Виконати раз на 10 хвилин.
 if (!timeaut10) {
 timeaut10 = 2; //timeaut10 = 2;
 Saved(); // зберегти налаштування
 }
```

```
 // Виконати раз на 10 хвилин.
 }
 // Виконати раз на 5 хвилини.
 break;
 case 7:
#if (VOLTIN == 1)
 // якщо зникла напруга від БП то
 if (vin_volt < ina.voltsec and ina.ampersec < 0) {
 vin_power_off--;
 if (!vin_power_off) {
 dcdc.Off();
 bitClear(flagsz, POWOFF); // Вимкнути. прапор живлення
 lcd.clear();
 lcd.print(F(txt4)); // "Power"
 print_mode(2); // "error"
 Saved(); // зберегти налаштування
 Speaker(500); // функція звукового сигналу (час у мілісекундах)
 }
 } else vin_power_off = 5; // 5 секунд
#endif
 break;
 }
 case 6:
 if (BitIsClear(flagsz, POWOFF)) {
#if (VOLTIN == 1)
 // якщо харчування від БП не надходить
 setCursory();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 2);

 if (vin_volt > ina.voltsec) {
 // якщо виникла напруга від БП
 vin_power_off = 5; // 5 секунд
 dcdc.setTok(10);
 Saved(); // зберегти налаштування
 bitSet(flagsz, POWOFF); // Прапор вимкнути. харчування
 Speaker(500); // функція звукового сигналу (час у мілісекундах)
 }
#endif
 }
 break;
 case 5:
 //перевірка те що силовий транзистор пробитий
 q1 = (dcdc.getTok() == 0 and ina.ampersec > 100) ? q1 + 1: 0;
 if (q1 > 2) bitWrite(pam.MyFlag, CHARGE, Q1_broken()); // якщо кількість перевірок перевищила 2, то запустити функцію
 break;
 case 4:
#if (FAN)
 tr_c = kul.fan(); // Читання температури / управління кулером;
#elif (SENSTEMP1)
 tr_c = read_tq1(); // Читання температури
#endif
// якщо підключений датчик температури і дозволено зменшувати струм при перевищенні температури транзистора
#if (SENSTEMP1 and GUARDTEMP)
 // якщо перевищена температура
 if (tr_c > vkr.service [DEGCUR]) bitSet (flagsz, TMPQ1); // перевищена температура Q1
 if (bitRead(flagsz, TMPQ1)) {
 if (tr_c > (vkr.service[DEGCUR] - 5)) dcdc.setCur_charge(constrain(dcdc.getCur_charge() - 20, 0, curr_charge_init)); // щось змен-
шуємо струм
 else {
 dcdc.setCur_charge(constrain(dcdc.getCur_charge() + 20, 0, curr_charge_init)); // щось збільшуємо струм
 if (tr_c> (vkr.service[DEGCUR] - 3)) bitClear(flagsz, TMPQ1);
 }
 }
#endif
break;
 case 3:
 // Захист від підвищеного струму та напруги
 if (ina.ampersec > (int16_t)(curr_charge_init * 1.2f) or (ina.voltsec > (uint16_t)(volt_charge_init * 1.03f) and ina.ampersec > 20)) {
 Light1(true); // увімкнути підсвічування дисплея
```

```
 if (dcdc.getTok()) {
 dcdc.Off();
 Speaker(200); // функція звукового сигналу (час у мілісекундах)
 } else {
 Speaker(300); // функція звукового сигналу (час у мілісекундах)
 resetFunc(); // Якщо струм або напруга заряду перевищить встановлений значить не працює блок регулювання та захисту та викликаємо reset (перезавантаження Ардуїно). Після перезавантаження робота продовжиться.
 }
 }
 break;
 case 2:
#if (VOLTIN == 1 and POWPIN == 2)
 PowerLight(); // перевіряє напругу від БП і якщо вона більш ніж на Акб включає пін 10 (за замовчуванням), для світлодіода
#endif
 Light1(false); // відключення підсвічування через 3 хвилини
#if (SENSTEMP2 == 2)
 bitWrite(flagsz, TAKBZ, Control_trAkb()); // контроль температури Акб з датчика NTC підключеного до пін А6 Ардуїно
#endif
 break;
 case 1:
#if (LOGGER == 1)
 logg--;
 if (!logg) {
 logg = LOGGTIME;
 Serial_out(ina.voltsec, ina.ampersec, tr_c, vin_volt, (add_charge ? pam.Ah_addcharge : pam.Ah_charge), 0); //----------
----------//---------- ----------//Serial_out(time_local, ina.voltsec, ina.ampersec, (add_charge ? pam .Ah_addcharge : pam.Ah_charge), (add_charge ? pam.Wh_addcharge : pam.Wh_charge), 0, tr_c, vin_volt); // функція відправлення значень до мережного порту
 }
#elif (LOGGER == 2)
 logg--;
 if (!logg) {
 logg = LOGGTIME;
 Serial_out(ina.voltsec, ina.ampersec, tr_c, (add_charge ? pam.Ah_addcharge : pam.Ah_charge)); //----------//--------
------- ----------//Serial_out(time_local, ina.voltsec, ina.ampersec); // функція відправлення значень до мережного порту
 }
#endif
 break;
 }
 time_millis--;
 }
 // якщо минула секунда
 }
 // цикл заряду
 dcdc.Off(); // Вимкнути заряд

// якщо підключений вентилятор
#if (FAN)
 kul.Off(); // Вимкнути вентилятор
#endif
#if (GUARDA0)
 digitalWrite_speed(PINA0, LOW);
#endif
 Saved(); // зберегти налаштування
}
```

DCDC

```
#include "FunctionLite.h"
#include "Arduino.h"

// клас управління DC-DC модулем
class DCDC {
public:
 // Список членів, доступних у програмі

 DCDC(void){};

 /// задати максимальні значення напруги та струму заряду
 void begin(uint16_t volt_charge, int16_t cur_charge) {
 _volt_charge = volt_charge;
 _cur_charge_max = cur_charge;
 }

 void start(void) {
 _cur_max = CURRMAXINT;
 _err_pred = 0;
 _Integral = 0.0;
 _smooth_start = true;
 _smooth_cur = 0;
 _smooth_latency = 0;
 //_kol = 0;
 // _tmp = micros();
 //_prd = 10000; // 3216 15822 31644
 }

#if (MCP4725DAC)
 // регулювання напруги та струму
 void Control() {
 if (micros() - _tmp >= 5000) {
 _tmp = micros();
 if (ina.isAlert() або abs(ina.amperms) >= _cur_max) _tok = constrain(_tok - 10, 0, 4095);
 else {
 int16_t volt_err = (int16_t)_volt_charge - (int16_t)ina.voltms;
 int16_t amp_err = _cur_charge_max - ina.amperms;
 int16_t err = (volt_err < amp_err)? volt_err: amp_err;

 float P = 0.001f*err;
 _Integral += 0.001f * err; // 0.001f * err;
 float D = (float) (err - _err_pred) * 5 * 0.0001; ///0.2 сек
 _err_pred = err;

 _tok = constrain (trunc (P + _Integral - D), 0, 4095); // Регулювання струму 0.012
 }
 dac.setVoltage(_tok);
 }
 }
#else

 // Регулювання напруги та струму заряду
 void Control() {
 //_kol++;
 if (abs(ina.amperms) >= _cur_max) _tok = constrain(_tok - 1, 0, 255);
 else {
 // Плавний пуск - плавне збільшення струму заряду
 if (_smooth_start) {
 if (_smooth_cur < _cur_charge_max) {
 if (++_smooth_latency > 100) {
 _smooth_cur + = 30;
 _smooth_latency = 0;
 }
 } else _smooth_start = false;
 }

 int16_t volt_err = (int16_t)_volt_charge - (int16_t)ina.voltms; // величина помилки за напругою
```

```
 int16_t amp_err = (_smooth_start ? _smooth_cur : _cur_charge_max) - ina.amperms; // величина помилки струму
 int16_t err = (volt_err < amp_err)? volt_err: amp_err; // регулюємо напругу чи струм

 float P = 0.001f*err;
 _Integral += P; // 0.001f * err;
 float D = (float) (err - _err_pred) * 5 * 0.0001;
 _err_pred = err;

 _tok = constrain (trunc (P + _Integral - D), 0, 255); // Регулювання струму 0.012
 }
 analogWrite_my(PWMCH, _tok);
 }

 void control_alert(void) {
 if (digitalRead_speed(4)) _tok = constrain(_tok - 1, 0, 255);
 else _tok = constrain(_tok + 1, 0, 255);
 analogWrite_my(PWMCH, _tok);
 }

#endif

// Регулювання напруги та струму розряду
#if (DISCHAR == 1)
 void Control_dich(void) {
 // Розряд
 int16_t volt_err = (int16_t)ina.voltms - (int16_t)pam.Volt_discharge; // величина помилки за напругою
 int16_t amp_err = pam.Current_discharge - abs(ina.amperms); // величина помилки струму
 int16_t err = (volt_err < amp_err) ? volt_err: amp_err; // регулюємо напругу чи струм
 _Integral += err * 0.001f;
 analogWrite_my(PWMDCH, (constrain(trunc(_Integral), 0, 255)));
 }
#endif
void Off(void) {
 _tok = 0;
 //_err_pred = 0;
 //_Integral = 0.0;
#if (MCP4725DAC)
 dac.setVoltage(_tok); // відключаємо заряд
#else
 digitalWrite_speed(PWMCH, LOW); // відключаємо заряд
#endif
#if (DISCHAR == 1)
 digitalWrite_speed(PWMDCH, LOW); // відключаємо розряд
#endif
 }

 void setVolt_charge(uint16_t v) {
 _volt_charge = v;
 }

 void setCur_charge(int16_t v) {
 _cur_charge_max = v;
 }

 uint16_t getVolt_charge(void) {
 return _volt_charge;
 }

 int16_t getCur_charge(void) {
 return _cur_charge_max;
 }


#if (MCP4725DAC)
 void setTok(uint16_t v) {
 _tok = v;
 }

 uint16_t getTok(void) {
```

```
 return _tok;
 }
#else
 void setTok(uint8_t v) {
 _tok = v;
 }

 uint8_t getTok(void) {
 return _tok;
 }
#endif
 /*
 uint16_t read_kol(void) {
 uint16_t k = _kol;
 _kol = 0;
 return k;
 }
*/
private:
 // Список членів для використання всередині класу
 uint16_t _volt_charge; // напруга заряду
 int16_t _cur_max; // граничний струм зарядника
 int16_t _cur_charge_max; // максимальний струм заряду
#if (MCP4725DAC)
 uint16_t _tok = 0; // значення ШІМ
#else
 uint8_t _tok = 0; // значення ШІМ
#endif
 int16_t _err_pred;
 float _Integral;
 bool _smooth_start; // Плавний пуск
 int16_t _smooth_cur; // Струм плавного пуску
 uint8_t _smooth_latency; // Затримка плавного пуску
 //uint32_t _tmp; // час

 // uint16_t _kol = 0; // кількість за секунду
};

extern DCDC dcdc;
DCDC dcdc = DCDC();
```

Функція розряду акумулятора

```
#if (DISCHAR == 1)
//Функція розряду акумулятора
void Discharge() {
 //виведення режиму, напруги та струму заряду/розряду
 PrintVA(pam.Volt_discharge, pam.Current_discharge, 0, 0, 1); // напруги та струму розряду
 printCykl(); // Виведення кількості циклів
 Freq(vkr.service[FREQDISCHAR]); // Встановити частоту роботи розрядного модуля (4 кГц за замовчуванням)
 const uint16_t s1 = ina.voltsec; // поточне вбрання акб;
 uint8_t tok = 1, n_disp = 0, ns_disp = 0;
 int8_t tiks = 0;
 bool dischar = true; // Розряд включений

 uint8_t time_millis = 0;

#if (LOGGER > 0)
 uint8_t logg=LOGGTIME; // період відправлення даних у мережевий порт
#endif
 Delay (3000);
 int8_t tr_c = 0; // значення температури
#if (FAN and KULDISCHAR == 2)
 digitalWrite_speed(PWMKUL, HIGH); // Примусове включення вентилятора при розряді
```

```
#endif

#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
#endif
 uint16_t time10 = 600; // Виконати раз на 600 сек (10 хв)
 disp.start(); // старт відліку часу одна секунда
 ina.start(2); // старт вимірів INA
 dcdc.start();
 // цикл розряду 35 годин максимум
 while (dischar and bitRead(pam.MyFlag, CHARGE) and pam.Time_discharge < 126000) {
 tiks = myTick(); // опитування кнопок та енкодера

 if (tiks) {
 // якщо кнопка натиснута
 switch (tiks) {
 case ENCHELD: // утримати енкодер
 case STOPHELD // утримати Стоп
 bitClear(pam.MyFlag, CHARGE); // завершити розряд
 break;
 case ENCRIGHT: // енкодер праворуч
 case PLUSCLICK: // натиснути плюс
 case ENCLEFT: // енкодер вліво
 case MINUSCLICK: // натиснути мінус
 n_disp++; // вікно відображення
 ns_disp = TIME_DISP; // час відображення сік
 lcd.clear();
 break;
 }
 Light1(true);
 }
 // читання та регулювання струму та напруги
 if (ina.sample()) {
 dcdc.Control_dich(); // контроль ШІМ розряду
#if (VOLTIN == 1)
 vin.sample(); // Читання напруги БП
#endif
 }

 if (disp.tick()) time_millis = 5;

 // якщо минула секунда
 if (time_millis) {
 switch (time_millis) {
 case 5:
 {
 int16_t ampsec = abs(ina.ampersec);
 pam.Ah_discharge += aw_ch((int32_t)ampsec); // Розрахунок ампер/годин
 pam.Wh_discharge += aw_ch(abs(ina.getPower())); // Розрахунок ват/годин
 pam.Time_discharge++; // Розрахувати час розряду

 if (ina.voltsec <= pam.Volt_discharge and ampsec <= pam.Current_discharge_min) dischar = false; // напруга зменшилася до вста-
новленого і струм зменшився до встановленого завершити розряд
 }
 break;
 case 4:
#if (KULDISCHAR == 0 або KULDISCHAR == 2)
 tr_c = read_tq1();
#endif
#if (FAN and KULDISCHAR == 1)
 tr_c = kul.fan(); // Читання температури / управління кулером;
#elif (SENSTEMP1)
 tr_c = read_tq1(); // Читання температури
#endif
 break;
 case 3:
 // Висновок на дисплей 1602
#if (DISPLAYy == 2)
 switch (n_disp) {
```

```
 // Висновок(вивід) на дисплей
case 0:
 // Екран 0 // Висновок поточних значень напруги, струму. Висновок Ватт-годин.
 Display_print(pam.Ah_discharge, pam.Time_discharge, map(ina.voltsec, pam.Volt_discharge, s1, 0, 100), false); // Висновок на дисплей
 break;
 case 1:
 // Екран 1 // Висновок середніх значень напруги, струму. Висновок Ватт-годин.
 setCursorx();
 print_mode(0);
 printSimb();
 printCykl();
 setCursory();
 PrintVA(0, 0, 0, pam.Wh_discharge, 2);
 lcd.print(map(ina.voltsec, pam.Volt_discharge, s1, 0, 100));
 lcd.print(F(txt_PRC));
 print_tr(tr_c); //температура
 if (ns_disp) ns_disp--;
 else n_disp = 0;
 break;
 // *Discharge C1 N1 *
 // * 45.25Wh 45% *
 default:
 n_disp = 0;
 break;
 }
// Висновок на дисплей 2004 або 1604
#elif (DISPLAYy == 4)
 // Висновок на дисплей
 switch (n_disp) {
 case 0:
 // Екран 0 висновок поточних значень
 Display_print(pam.Ah_discharge, pam.Wh_discharge, pam.Time_discharge, tr_c, map(ina.voltsec, pam.Volt_discharge, s1, 0, 100));
 break;
 case 1:
 // Екран 1
 setCursorx();
 print_mode(0);
 printSimb();
 printCykl();
 setCursory();
 print_tr(tr_c); //температура
 if (ns_disp) ns_disp--;
 else n_disp = 0;
 break;
 default:
 n_disp = 0;
 break;
 }
#endif
 break;

 case 2:
#if (POWPIN == 1)
 Relay(); // Функція управління реле підключення до мережі 220В
#endif
 // Вимірювання напруги блоку живлення
#if (VOLTIN == 1 and POWPIN == 2)
 PowerLight(); // перевіряє напругу від БП і якщо вона більш ніж на Акб включає пін 10 (за замовчуванням), для світлодіода
#endif
 Light1(false); // відключення підсвічування через 3 хвилини
 break;

 case 1:
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
#if (LOGGER == 1)
 logg--;
```

```
 if (!logg) {
 logg = LOGGTIME;
 Serial_out(ina.voltsec, ina.ampersec, tr_c, 0, pam.Ah_discharge, 0); //----------------------//---------------------------- -
//Serial_out(pam.Time_discharge, ina.voltsec, ampsec, pam.Ah_discharge, pam.Wh_discharge, 0, 0, 0); // функція відправлення зна-
чень до мережного порту
 }
#endif
#if (LOGGER == 2)
 logg--;
 if (!logg) {
 logg = LOGGTIME;
 Serial_out(ina.voltsec, ina.ampersec, tr_c, pam.Ah_discharge); //--------------------------------//---------------- -------
//Serial_out(pam.Time_discharge, ina.voltsec, ampsec); // функція відправлення значень до мережного порту
 }
#endif

 time10--;
 // Виконати раз на 10 хв
 if (!time10) {
 time10 = 600;
 Saved(); // зберегти налаштування
 }
 break;
 }
 time_millis--;
 }
 // Виконати раз на секунду
 }
// цикл розряду
 // Light1 (true);
 dcdc.Off(); // відключаємо розряд
#if (KULDISCHAR)
 kul.Off(); // Вимкнути вентилятор
#endif
#if (GUARDA0)
 digitalWrite_speed(PINA0, LOW);
#endif
 Saved(); // зберегти налаштування
}
// Функція розряду акумулятора
#endif


#if (RESIST == 1)
// Функція виміру опору
void Resist() {
 if (ina.voltsec < 2000) {
 print_mode(2); // "error"
 Delay (1000);
 return;
 }
 Freq(vkr.service[FREQDISCHAR]); // Встановити частоту роботи розрядного модуля (4 кГц за замовчуванням)
 uint16_t U[2] = {0};
 int16_t I[2] = {0};
 uint8_t timer = 8; // Час першого виміру сек.
 int8_t tiks = 0;
 Fixcurrent(400); // встановити струм розряду ма
 disp.start(); // старт відліку часу одна секунда
 ina.start(2); // старт вимірів INA
 //виконувати 10 сек
 do {
 tiks = myTick();
 ina.sample();
 if (tiks == ENCHELD or tiks == STOPCLICK) {
 digitalWrite_speed(PWMDCH, LOW); // Вимкнути розряд
 return;
 }
 if (disp.tick()) {
 // якщо пройшла секунда виведення на дисплей
```

```
 setCursory();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 2); // * 12.36V 3.55A *
 timer--;
 }
 } while (timer);
 I [0] = abs (ina.ampersec);
 U[0] = ina.voltsec;
 uint32_t t = millis();
 digitalWrite_speed(PWMDCH, HIGH); // увімкнути максимальний струм
 // очікування
 while (millis() - t < 1000) ina.sample(); // Очікування другого виміру, мілісек.
 I[1] = abs(ina.ampersec); // Читання струму ina.amperms
 U[1] = ina.voltsec; // Читання напруги ina.voltms
 digitalWrite_speed(PWMDCH, LOW); // Вимкнути розряд
 setCursory();
 PrintVA(U[1], I[1], 0, 0, 2); // * 12.36V 3.55A *
 Delay (3000); // очікування 3 сек
 uint16_t r = (U[0] - U[1]) * 100000/(I[1] - I[0]); // розрахувати внутрішній опір акумулятора
 setCursory();
 lcd.print(((float)r/100), 1);
 lcd.print(F("mOm "));
 if(r) {
 pam.Resist_1 = r;
 pam.Resist_2 = (uint16_t)((uint32_t)bat.Volt(3) * 100 / r); // Розрахунок пускового струму - напругу ділити на опір
 lcd.print(pam.Resist_2);
 lcd.write(65); // A
 }
 ClearStr(5);
 Delay (7000); // очікування
}
#endif
```

Вивід інформації на дисплей

```
/ / Висновок на екран в float - напруга, струм, ампер-годинник, ват-годинник, кільк. розрядів після коми.
void PrintVA(uint16_t volt, int16_t current, int32_t Ah, int32_t Wh, uint8_t x) {
 if (volt) {
 lcd.print(((float)volt/1000), x); // напруга
 lcd.print(F("V"));
 }
 if (current) {
 lcd.print(((float)current / 1000), x); // Струм
 lcd.print(F("A"));
 }
 if (Ah) {
 lcd.print(((float)Ah/100000), x); // ампер-годинник
 lcd.print(F("Ah")); // Ah
 }
 if (Wh) {
 lcd.print(((float)Wh/100000), x); // ват-годинник
 lcd.print(F("Wh")); //Wh
 }
}

void pr_tm(uint16_t tm) {
 if (tm < 10) lcd.print(0);
 lcd.print(tm);
}

// функція розрахунку поточного часу та виведення на дисплей
void Print_time(uint32_t time_real, uint8_t x, uint8_t y) {
 lcd.setCursor(x, y);
 pr_tm((uint16_t)(time_real / 3600ul)); // годинник
 lcd.write(58);
 uint32_t t = (time_real % 3600ul); // хвилини
 pr_tm((uint16_t)(t/60ul));
 lcd.write(58);
 pr_tm((uint16_t)(t % 60ul)); // секунди
```

```
}

void setCursorx(void) {
 lcd.setCursor(0, 0);
}
void setCursory(void) {
 lcd.setCursor(0, 1);
}
void printSimb(void) {
 lcd.write(PROBEL);
}

void print_tr(int8_t tr) {
 lcd.print(tr);
 lcd.print(F(txt_C));
}

// функція виведення на дисплей
#if ((DISPLAYx == 16 and DISPLAYy == 2) or (DISPLAYx == 20 and DISPLAYy == 2))
void Display_print(int32_t Ah, uint32_t time_real, uint8_t prc, bool add) {
 static bool akb = true;
 // 0 рядок
 setCursorx();
 if (ina.voltsec < 10000) lcd.print(0);
 PrintVA(ina.voltsec, 0, Ah, 0, 2); // Висновок напруги та Ач
 /*
 lcd.setCursor(DISPLAYx - 4, 0);
 lcd.print(dcdc.read_kol());
 printSimb();
*/
 lcd.setCursor(DISPLAYx - 1, 0);
 lcd.print((char)pgm_read_byte(&regimr[(add ? 3 : pam.Mode)])); // Висновок режиму роботи
 // 1 рядок
 setCursory();
 PrintVA(0, ((pam.Mode == 4) ? abs(ina.ampersec) : ina.ampersec), 0, 0, ((abs(ina.ampersec) < 10000) ? 3 : 2));
 Print_time(time_real, 7, 1);
 lcd.setCursor(DISPLAYx - 1, 1);
 lcd.write((akb ? map(prc, 0, 100, 0, 6) : ((pam.Mode == 4) ? 32 : 7))); // Знак акб: пробіл - знак заряду
 akb = akb;
}
// * 14.81V 0.01Ah Ch * 0
// *10.55A 01:01:01 * 1

// 1 - використовується дисплей 20 * 4
#elif (DISPLAYx == 20 and DISPLAYy == 4)
// функція виведення на екран 20*4
void Display_print(int32_t Ah, int32_t Wh, uint32_t time_real, int8_t tr, uint8_t prc) {
 static bool akb = true;
 // 0 рядок
 setCursorx();
 print_mode(1); // Висновок типу Акб // Тип Акб
 printSimb(); // пробіл
 print_Capacity();
 print_mode(4); // 12.6V
 // 1 рядок
 Print_time(time_real, 0, 1); // час
 printSimb(); // пробіл
 print_mode(0); // Режим роботи
 // 2 рядок
 lcd.setCursor(0, 2);
 PrintVA(0, 0, Ah, Wh, 2); // ампер-годинник ват-годинник
 lcd.print(prc); // Відсоток заряду Акб
 lcd.print(F(txt_PRC));
 // 3 рядок
 lcd.setCursor(0, 3);
 if (ina.voltsec < 10000) lcd.print(0);
 PrintVA(ina.voltsec, 0, 0, 0, 2); // вивести Вольт
 PrintVA(0, ((pam.Mode == 4) ? abs(ina.ampersec) : ina.ampersec), 0, 0, ((abs(ina.ampersec) < 10000) ? 3 : 2)); // вивести Ампер
 print_tr(tr); //температура
```

```
 lcd.setCursor(DISPLAYx - 1, 3);
 lcd.write((akb ? map(prc, 0, 100, 0, 6) : ((pam.Mode == 4) ? 32 : 7))); // знак акб - знак заряду - пробіл
 akb = akb;
}
// * PB Ca/Ca 60Ah 12.6V * 0
// *01:01:01 Zaryad>Doza* 1
// *60.56Ah 220.56Wh 56%* 2
// *14.81V 6.555A 35C ~* 3


// 1 - використовується дисплей 16 * 4
#elif (DISPLAYx == 16 and DISPLAYy == 4)
// функція виведення на екран 20*4
void Display_print(int32_t Ah, int32_t Wh, uint32_t time_real, int8_t tr, uint8_t prc) {
 static bool akb = true;
 // 0 рядок
 setCursorx();
 print_Capacity();
 lcd.print(((float)bat.Volt(1) / 100), 1); // напруга
 lcd.print(F("V")); // V
 print_mode(0); // Режим роботи
 // 1 рядок
 Print_time(time_real, 0, 1); // час
 printSimb(); // пробіл
 lcd.print(prc); // Відсоток заряду Акб
 lcd.print(F(txt_PRC));
 print_tr(tr); //температура
 // 2 рядок
 lcd.setCursor(0, 2);
 PrintVA(0, 0, Ah, Wh, 2); // ампер-годинник ват-годинник
 // 3 рядок
 lcd.setCursor(0, 3);
 if (ina.voltsec < 10000) lcd.print(0);
 PrintVA(ina.voltsec, 0, 0, 0, 2); // вивести Вольт
 PrintVA(0, ((pam.Mode == 4) ? abs(ina.ampersec) : ina.ampersec), 0, 0, ((abs(ina.ampersec) < 10000) ? 3 : 2)); // вивести Ампер
 lcd.setCursor(DISPLAYx - 1, 3);
 lcd.write((akb ? map(prc, 0, 100, 0, 6) : ((pam.Mode == 4) ? 32 : 7))); // знак акб - знак заряду - пробіл
 akb = akb;
}
// *60Ah 12.6V Zarya*
// *01:01:01 52% 35C*
// *60.56Ah 220.56Wh*
// *14.81V 6.555A ~*

#endif

// вивести на дисплей - вкл/вимк
void PrintOnOff(bool i) {
 lcd.print((i ? F(txt13) : F(txt14)))); // вкл / вимк
}

// виведення на дисплей > або пробіл
void Write_x(bool x) {
 lcd.write((x ? 62 : 32)); // > or пробіл
}

// Висновок на дисплей: 0-режим роботи, 1-тип акб
void print_mode(uint8_t i) {
 switch (i) {
 case 0: printFromPGM((int)(&modeTxt[pam.Mode])); break;
 case 1: printFromPGM((int)(&typeAkb[pam.typeAkb])); break;
 case 2: lcd.print(F(txt21)); break; // "error"
 case 3: printFromPGM((int)(&menuTxt[5])); break; // "Settings"
 case 4:
 lcd.print(((float)bat.Volt(0) / 1000), 1); // напруга
 lcd.write(86); // V
 break;
 }
}
```

```
void print_Capacity(void) {
 lcd.print(pam.Capacity);
 lcd.print(F("Ah")); // Ah
}

void printCykl(void) {
 lcd.write(67); // C
 lcd.print(pam.Round);
}

// Функція виведення напруги акб та кількість банок
void Vout(void) {
 print_mode(4); // 12.6V
 lcd.write(40); // (
 lcd.print(pam.Voltage);
 lcd.write(41); //)
 ClearStr(4); // Очистити поле
 lcd.setCursor(3, 0);
}

// функція очищення рядка дисплея (і символів)
void ClearStr(uint8_t i) {
 while (-i) printSimb(); // Очистити поле
}
```

Налаштування INA 226

```
#include "Arduino.h"
#include <microWire.h>

/*
Конструктор:
 INA226 ina226 (Адреса на шині I2c)
 INA226 ina226 (0x41); // Шунт та макс. стандартний струм, адреса 0x41 - підійде для декількох модулів

Методи:
 bool begin(); // Ініціалізація модуля та перевірка присутності, поверне false якщо INA226 не знайдена
 void sleep(true/false); // Вмикання та вимикання режиму низького енергоспоживання, залежно від аргументу
 void setAveraging(avg); // Установка кількості усереднень вимірів (див. таблицю нижче)
 void setSampleTime(ch, time); // Установка часу вибірки напруги та струму (INA226_VBUS / INA226_VSHUNT), за замовчу-
ванням INA226_CONV_1100US

 float getShuntVoltage(); // Прочитати напругу на шунті
 float getVoltage(); // Прочитати напругу
 float getCurrent(); // Прочитати струм
 float getPower(); // Прочитати потужність

 uint16_t getCalibration(); // Прочитати калібрувальне значення (після старту розраховується автоматично)
 void setCalibration(calibration value); // Записати калібрувальне значення (можна зберігати його в EEPROM)
 void adjCalibration(calibration offset); // Підкрутити значення калібрування на вказане значення (можна змінювати на ходу)


*/

/* Public-визначення (константи) */
#define INA226_VBUS true // Канал АЦП, що вимірює напругу шини (0-36в)
#define INA226_VSHUNT false // Канал АЦП, що вимірює напругу на шунті

// Час вибірки (накопичення сигналу для оцифрування)
#define INA226_CONV_140US 0b000
#define INA226_CONV_204US 0b001
```

```
#define INA226_CONV_332US 0b010
#define INA226_CONV_588US 0b011
#define INA226_CONV_1100US 0b100
#define INA226_CONV_2116US 0b101
#define INA226_CONV_4156US 0b110
#define INA226_CONV_8244US 0b111

// Вбудоване усереднення (пропорційно збільшує час оцифрування)
#define INA226_AVG_X1 0b000
#define INA226_AVG_X4 0b001
#define INA226_AVG_X16 0b010
#define INA226_AVG_X64 0b011
#define INA226_AVG_X128 0b100
#define INA226_AVG_X256 0b101
#define INA226_AVG_X512 0b110
#define INA226_AVG_X1024 0b111

/* Private-визначення (адреси) */
#define INA226_CFG_REG_ADDR 0x00
#define INA226_SHUNT_REG_ADDR 0x01
#define INA226_VBUS_REG_ADDR 0x02
#define INA226_POWER_REG_ADDR 0x03
#define INA226_CUR_REG_ADDR 0x04
#define INA226_CAL_REG_ADDR 0x05

#define INA226_REG_MASKENABLE 0x06
#define INA226_REG_ALERTLIMIT 0x07

#define INA226_BIT_AFF 0x0010 // (біт 4) // Читання регістру спрацьовування Alert

#define INA226_BIT_SOL (1 << 15) //0x8000 // (біт 15)Shunt Voltage Over-Voltage. Установка 1 цього біта конфігурує установку ніжки Alert, якщо виміряна напруга шунта перевищила значення, запрограмоване Alert Limit Register.
//#define INA226_BIT_SUL 0x4000 // (біт 14)Shunt Voltage Under-Voltage. Установка 1 цього біта конфігурує установку ніжки Alert, якщо виміряна напруга шунта впала нижче значення, запрограмованого в Alert Limit Register.
//#define INA226_BIT_BOL 0x2000 // (біт 13) Bus Voltage Over-Voltage. Установка 1 цього біта конфігурує установку ніжки Alert, якщо виміряна напруга VBUS перевищила значення, запрограмоване в Alert Limit Register.
//#define INA226_BIT_BUL 0x1000 // (біт 12) Bus Voltage Under-Voltage. Установка 1 цього біта конфігурує установку ніжки Alert, якщо виміряна напруга VBUS впала нижче значення, запрограмованого в Alert Limit Register.
//#define INA226_BIT_POL 0x0800 // (біт 11)Power Over-Limit. Установка 1 цього біта конфігурує установку ніжки Alert, якщо обчислена потужність після вимірювання напруги VBUS перевищила значення, запрограмоване Alert Limit Register.
//#define INA226_BIT_CNVR 0x0400 // (біт 10) Conversion Ready. Установка 1 цього біта конфігурує установку ніжки Alert, коли встановиться Conversion Ready Flag (біт 3), показуючи тим самим готовність до наступного перетворення.
//#define INA226_BIT_AFF 0x0010 // (біт 4)Alert Function Flag. Хоча будь-якої миті часу ніжкою Alert може відстежуватися лише одна функція Alert, ніжкою Alert також може відстежуватися і прапор Conversion Ready. Читання Alert Function Flag після події alert дає можливість користувачеві визначити, чи функція Alert була джерелом цієї події.
//#define INA226_BIT_CVRF 0x0008 // (біт 3)Conversion Ready Flag, щоб допомогти координувати одноразові перетворення (або triggered-перетворення). Біт Conversion Ready Flag встановиться після завершення всіх перетворень, усереднення та множення.
//#define INA226_BIT_OVF 0x0004 // (біт 2) Math Overflow Flag. Цей біт встановиться у 1, якщо арифметична операція призвела до помилки переповнення. Це показує, що дані струму та потужності можуть бути недостовірними.
//#define INA226_BIT_APOL 0x0002 // (біт 1)Alert Polarity bit. Цей біт встановлює полярність активації ніжки Alert. 1 = інвертована полярність (відкритий колектор з активною балкою 1). 0 = нормальна полярність (відкритий колектор з активним лог. 0, налаштування за замовчуванням).
//#define INA226_BIT_LEN 0x0001 // (біт 0)Alert Latch Enable - Конфігурує роздільну здатність замикання функції ніжки Alert і біт Alert Flag. 1 = замикання дозволено (режим Latch). 0 = прозорий режим (режим Transparent, стандартна настройка).



class INA226 {
public:

 INA226(uint8_t address)
 : _iic_address(address) {}

 // Ініціалізація та перевірка
 bool begin() {
 Wire.begin(); // Ініціалізація шини I2c
 if (!testConnection()) return false; // Перевірка присутності
```

```
 writeRegister(INA226_REG_MASKENABLE, INA226_BIT_SOL); // Сигнал Alert спрацює при перевищенні напруги на шунті 0,081 Вольт. При шунті 0,01 Ом струму спрацьовування 0,081/0,01=8,1А
 writeRegister(INA226_REG_ALERTLIMIT, 32400); //0,081/0,0000025f обмеження по напрузі на шунті 32400 * 0.0025 = 81мВ
 return true; // Повернути true якщо всі ок
 }

 // Установка/зняття режиму сну
 void sleep(bool state) {
 uint16_t cfg_register = readRegister(INA226_CFG_REG_ADDR) & ~(0b111); // Прочитати конф. регістр і стерти біти режиму
 writeRegister(INA226_CFG_REG_ADDR, cfg_register | (state ? 0b000 : 0b111)); // Записати нове значення конф. регістру з вибраним режимом
 }

 // Установка калібрувального значення
 void setCalibration(uint16_t cal) {
 writeRegister(INA226_CAL_REG_ADDR, cal); // Пишемо значення в регістр калібрування
 _cal_value = cal; // Оновлюємо внутрішню змінну
 }

 // Підстроювання калібрувального значення
 void adjCalibration(int16_t adj) {
 setCalibration(getCalibration() + adj); // Читаємо та модифікуємо значення
 _cal_value = _cal_value + adj; // Оновлюємо внутрішню змінну
 }

 // Читання калібрувального значення
 uint16_t getCalibration(void) {
 _cal_value = readRegister(INA226_CAL_REG_ADDR); // Оновлюємо внутрішню змінну
 return _cal_value; // Повертаємо значення
 }

 // Установка вбудованого усереднення вибірок
 void setAveraging(uint8_t avg) {
 uint16_t cfg_register = readRegister(INA226_CFG_REG_ADDR) & ~(0b111 << 9); // Читаємо конф. регістр, скинувши біти AVG2-0
 writeRegister(INA226_CFG_REG_ADDR, cfg_register | avg << 9); // Пишемо нове значення конф. регістр
 }

 // Встановлення роздільної здатності для вибраного каналу
 void setSampleTime(bool ch, uint8_t mode) {
 uint16_t cfg_register = readRegister(INA226_CFG_REG_ADDR); // Читаємо конф. регістр
 cfg_register &= ~((0b111) << (ch ? 6 : 3)); // Скидаємо потрібну пачку біт, залежно від каналу
 cfg_register |= mode << (ch ? 6 : 3); // Пишемо потрібну пачку біт, залежно від каналу
 writeRegister(INA226_CFG_REG_ADDR, cfg_register); // Пишемо нове значення конф. регістру
 }

 // Читання напруги на шунті
 int8_t getShuntVoltage(void) {
 // setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
 int16_t value = readRegister(INA226_SHUNT_REG_ADDR); // Читання регістра напруги шунта // 0x7FFF (максимальне значення int16_t)
 return (int8_t)(value * 0.0025f); //LSB = 2.5uV = 0.0000025V, множимо та повертаємо // return value * 0.0000025f;
 }

 // Читання напруги на шунті з регістру
 int16_t getShuntHex(void) {
 // setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
 int16_t value = readRegister(INA226_SHUNT_REG_ADDR); // Читання регістра напруги шунта // 0x7FFF (максимальне значення int16_t) +- 32767
 return value;
 }

 // Читання напруги
 uint16_t getVoltage(void) {
 uint16_t value = readRegister(INA226_VBUS_REG_ADDR); // Читання регістра напруги
 return (uint16_t)trunc(value * _vkoof); // LSB = 1.25mV = 0.00125V, Зсуваємо значення до 12 біт і множимо // return value * 0.00125f;
 }
```

```
// Читання напруги з регістру
uint16_t getVoltageHex(void) {
uint16_t value = readRegister(INA226_VBUS_REG_ADDR); // Читання регістра напруги 0x7FFF (максимальне значення uint16_t) + 32767 Діапазон повної шкали 40.96V
return value; // Перетворити на напругу * 1.25f // Перетворити напр. у реєстр. IHA * 0.8f
}

// Читання струму
int16_t getCurrent(void) {
//setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
int16_t value = readRegister(INA226_CUR_REG_ADDR); // Читання регістру струму
return (int16_t)trunc(value * _current_lsb); //LSB розраховується з урахуванням макс. очікуваного струму, множимо і повертаємо ret urn value *_current_lsb;
}

// Читання струму
int16_t getCurrentHex(void) {
setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
int16_t value = readRegister(INA226_CUR_REG_ADDR); // Читання регістру струму
return value;
}

// Читання потужності
int32_t getPower(void) {
//setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
int16_t value = readRegister(INA226_POWER_REG_ADDR); // Читання регістру потужності
return (int32_t)trunc(value * _power_lsb); // LSB у 25 разів більше за LSB для струму, множимо повертаємо // return value * _power_lsb;
}

// Читання регістру спрацьовування Alert
bool isAlert(void) {
return ((readRegister(INA226_REG_MASKENABLE) & INA226_BIT_AFF) == INA226_BIT_AFF);
}

//********* обчислення середнього **********
uint16_t voltms = 0;
int16_t amperms = 0;
uint16_t voltsec = 0;
int16_t ampersec = 0;

/*
час вибірки (накопичення сигналу для оцифрування)
INA226_CONV_140US // 140 мкс
INA226_CONV_204US // 204 мкс
INA226_CONV_332US // 332 мкс
INA226_CONV_588US // 588 мкс
INA226_CONV_1100US // 1100 мкс
INA226_CONV_2116US // 2116 мкс
INA226_CONV_4156US // 4156 мкс
INA226_CONV_8244US // 8244 мкс
*/

void start(uint8_t i) {
_r_shunt = (float) vkr.shunt/100000; // Опір шунта
_i_max = 0.08192f/_r_shunt; // максимальний струм залежно від опору шунта
_vkoof = INAVKOOFV; // Коефіцієнт напруги INA226
_ohms=(float)vkr.Ohms/1000; // опір ланцюга від INA до акумулятора Ом
calibrate(); // Розрахунок калібрувального значення та ініціалізація
setCalibration(_cal_value); // Примусове оновлення калібрування (на випадок раптового ребуту INA226)
_vs = 0;
_as = 0;
switch (i) {
case 1:
// Параметри для заряду
setSampleTime(INA226_VBUS, INA226_CONV_140US); // час вибірки напруги // 140 * 16 * 2 = 4480 // 2116 * 2 = 4232 // 332 * 16 * 2 = 10624 мкс
setSampleTime(INA226_VSHUNT, INA226_CONV_140US); // час вибірки струму
```

```
 // setAveraging (INA226_AVG_X4); // 4х кратне усереднення, за умовчанням усереднення немає (1, 4, 16, 64, 128, 256, 512, 1024).
 _prd = 300; // 780/846 Гц
 break;
 case 2:
 // Параметри для розряду
 setSampleTime(INA226_VBUS, INA226_CONV_588US); // час вибірки напруги // 140 * 16 * 2 = 4480 // 2116 * 2 = 4232 // 332 * 16 * 2 = 10624 мкс
 setSampleTime(INA226_VSHUNT, INA226_CONV_588US); // час вибірки струму
 setAveraging(INA226_AVG_X4); // 4х кратне усереднення, за умовчанням усереднення немає (1, 4, 16, 64, 128, 256, 512, 1024).
 _prd = 4704; // 212 Гц
 break;
 }
 _tmp = micros();
 }

 bool sample(void) {
 bool fg = false;
 if (micros() - _tmp >= _prd) {
 _tmp = micros();
 fg = true;
 amperms = getCurrent(); // Читання струму -32767mA - +32767mA max
 voltms = getVoltage() - (int16_t)(amperms * _ohms); // vkr.Ohms / 1000 // читання напруги 0 - 65535mV max з коригуванням

 _vs += ((((int32_t)voltms << 4) - _vs) >> 4); // ковзне середнє - згладити в 16 разів
 voltsec = (uint16_t) (_vs >> 4);
 _as += ((((int32_t)amperms << 4) - _as) >> 4); // ковзне середнє - згладити в 16 разів
 ampersec = (int16_t) (_as >> 4);
 }
 return fg;
 }

 // функція INA226 error
 void error(void) {
 lcd.clear();
 lcd.print(F(txt6)); // INA226 error
 delay(3000);
 }


private:
 uint8_t _iic_address = 0x40; // Адреса на шині I2c
 float _r_shunt; // Опір шунта = 0.1f
 float _i_max; // Макс. очікуваний струм = 0.8f

 float _current_lsb = 0.0; // LSB для струму
 float _power_lsb = 0.0; // LSB для потужності
 uint16_t _cal_value = 0; // Калібрувальне значення

 // Запис 16-бітного регістру INA226
 void writeRegister(uint8_t address, uint16_t data) {
 Wire.beginTransmission(_iic_address); // Починаємо передачу
 Wire.write(address); // Відправляємо адресу
 Wire.write(highByte(data)); // Відправляємо старший байт
 Wire.write(lowByte(data)); // Відправляємо молодший байт
 Wire.endTransmission(); // Закінчуємо передачу
 }

 // Читання 16-бітного регістру INA226
 uint16_t readRegister(uint8_t address) {
 Wire.beginTransmission(_iic_address); // Починаємо передачу
 Wire.write(address); // Відправляємо адресу
 Wire.endTransmission(); // Закінчуємо передачу
 Wire.requestFrom(_iic_address, (uint8_t)2); // Запитуємо 2 байти
 return Wire.read() << 8 | Wire.read(); // Клеїмо і повертаємо результат
 }

 // Перевірка присутності
```

```
 bool testConnection(void) {
 Wire.beginTransmission(_iic_address); // Починаємо передачу
 return (bool)!Wire.endTransmission(); // Відразу закінчуємо, інвертуємо результат
 }

 // Процедура розрахунку калібрувального значення та ініціалізації
 void calibrate(void) {
 writeRegister(INA226_CFG_REG_ADDR, 0x8000); // Примусове скидання
 _current_lsb = _i_max/32.768f; // Розрахунок LSB для струму (див. доку INA226)
 _power_lsb = _current_lsb * 25.0f; // Розрахунок LSB для потужності (див. доку INA226)
 _cal_value = trunc(0.00512f / (_i_max / 32768.0f * _r_shunt)); // Розрахунок калібрувального значення (див. доку INA226)
 setCalibration(_cal_value); // Записуємо стандартне калібрувальне значення
 }

 //********* обчислення середнього **********
 uint16_t _prd; // період вимірів у мкс
 uint32_t _tmp; // час
 float _ohms;
 int32_t _vs = 0;
 int32_t _as = 0;
 float _vkoof = 1.25f;
};
```

Налаштування кулера

```
// Читання температури силового транзистора
int8_t read_tq1(void) {
 int8_t tr = 0; // значення температури, що повертається з функції
 // якщо підключено датчик температури
#if (SENSTEMP1 == 1)
 // Перевірка DS18B20
 if (tr_Q1.online()) {
 if (tr_Q1.readTemp()) tr = (int8_t)tr_Q1.getTempInt(); // Прочитати температуру з датчика. [true якщо успішно] // отримати значення температури в int
 tr_Q1.requestTemp(); // Запросити нове перетворення температури
 }

#elif (SENSTEMP1 == 2)
 // якщо підключено датчик температури NTC
 tr = tr_Q1.getTempInt(); // Читання температури з датчика 1
#endif
 return tr; // Повернути значення температури
}

// клас управління кулером
class KULER {
public:
 // Список членів, доступних у програмі
 KULER (void) {};
 // ініціалізація
 void begin(void) {
 // Піни D9 та D10 - 30 Гц для вентилятора
 TCCR1A = 0b00000001; // 8bit
 TCCR1B = 0b00000101; // x1024 phase correct
 _kul = 0; // значення ШІМ кулера
 _kulb = false; // кулер вкл/вимк
 _curfanOn = (int16_t)vkr.service[CURFANON] * 100; // Значення струму в Амперах, при якому включається вентилятор. (до 25,5A) define в 2_menu.h
 _curfanOff = (int16_t)vkr.service[CURFANOFF] * 100; // значення струму в Амперах, при якому відключається вентилятор. (До 25,5A)
```

```
 _curfanMax=(int16_t)vkr.service[CURFANMAX]*100; // Значення струму в Амперах для максимальних обертів вентилятора ШІМ. (До 25,5А)
 }

 int8_t fan(void) {
 int8_t tr = read_tq1(); // Читання температури силового транзистора
#if (FAN)
 int16_t = abs(ina.ampersec); // Читання струму
 switch (vkr.service[FANMODE]) {
 case 1:
 // управління кулером ШІМ
 if (tr> = vkr.service [DEGMAX]) _kul = 255;
 else _kul = tr? ((tr >= vkr.service[DEGON]) ? (_kul ? map(tr, vkr.service[DEGOFF], vkr.service[DEGMAX], 70, 255) : 150) : _kul) : _curfanOn) ? (_kul ? map(amp, _curfanOff, _curfanMax, 70, 255) : 150) : _kul);
 // якщо кулер увімкнено
 if (_kul) {
 _kul = tr? ((tr <= vkr.service[DEGOFF]) ? 0 : _kul) : ((amp <= _curfanOff) ? 0 : _kul);
 }
 analogWrite_my(PWMKUL, _kul);
 break;

 case 2:
 // управління кулером вкл/вимк
 // якщо кулер увімкнено
 if (_kulb) {
 _kulb = tr? ((tr <= vkr.service[DEGOFF]) ? false : _kulb) : ((amp <= _curfanOff) ? false : _kulb);
 } else {
 // якщо кулер вимкнено
 _kulb = tr? ((tr >= vkr.service[DEGON]) ? true : _kulb) : ((amp >= _curfanOn) ? true : _kulb);
 }
 digitalWrite_speed(PWMKUL, _kulb);
 break;
 }
#endif
 return tr; // Повернути значення температури
 }

 void Off(void) {
 _kul = 0;
 _kulb = false;
 digitalWrite_speed(PWMKUL, LOW);
 }

private:
 // Список членів для використання всередині класу
 uint8_t _kul; // значення ШІМ кулера
 bool _kulb; // кулер вкл/вимк
 int16_t _curfanOn; // Значення струму в Амперах, при якому включається вентилятор. (до 25,5А) define в 2_menu.h
 int16_t _curfanOff; // значення струму в Амперах, при якому відключається вентилятор. (До 25,5А)
 int16_t _curfanMax; // Значення струму в Амперах для максимальних обертів вентилятора ШІМ. (До 25,5А)
};
extern KULER kul;
KULER kul = KULER();
```

Бібліотека MCP4725my

```
// бібліотека до роботи з ЦАП MCP4725
// просто відправляє значення 0-4095 напруги MCP4725

#include <microWire.h>

class MCP4725 {
public:
 // Список членів, доступних у програмі

 // ініціалізація (номер аналогового піна)
 MCP4725(uint8_t addr) : _address(addr) {
 Wire.begin();
 }
```

```
 // Запис 16-бітного регістру MCP4725
 void setVoltage(uint16_t data) {
 Wire.beginTransmission(_address); // Починаємо передачу
 Wire.write(0b01000000); // передаємо контрольний байт (010-Sets in Write mode)
 Wire.write((uint8_t)(data >> 4)); // Upper data bits (D11.D10.D9.D8.D7.D6.D5.D4) Відправляємо старший байт
 Wire.write((uint8_t)((data % 12) << 4)); // Lower data bits (D3.D2.D1.D0.x.x.x.x)); // Відправляємо молодший байт
 Wire.endTransmission(); // Закінчуємо передачу
 }

 // Перевірка присутності
 bool testConnection(void) {
 Wire.beginTransmission(_address); // Починаємо передачу
 return (bool)!Wire.endTransmission(); // Відразу закінчуємо
 }

private:
 const uint8_t_address;
};
```

Функція захисту

```
// функція якщо пробитий силовий транзистор
bool Q1_broken() {
 Light1(true); // увімкнути підсвічування
 bitWrite(vkr.GlobFlag, TRQ1, false);
 dcdc.Off();
#if (GUARDA0)
 digitalWrite_speed(PINA0, LOW); // Розблокувати модуль захисту
#endif
#if (PROT == 1)
 digitalWrite_speed(PROTECT, HIGH); //закрити захисний транзистор
#endif
#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, LOW); // Вимкнути мережу 220В
#endif

 Saved(); // зберегти налаштування
 lcd.clear();
 lcd.print(F(txt24)); //"!!-BROKEN Q1-!!"

 do {
 Speaker(1000); // функція звукового сигналу (час у мілісекундах)
 } while (! Delay (10000)); // Чекати 10 секунд

 bitWrite(vkr.GlobFlag, TRQ1, Choice(60, false)); // при натисканні ОК записати на згадку про те, що транзистор справний, //
при натисканні стоп (через 60 секунд) записати на згадку про те, що транзистор пробитий
 Saved(); // зберегти налаштування
#if (PROT == 1)
 if (BitIsClear(vkr.GlobFlag, TRQ1)) digitalWrite_speed(PROTECT, LOW); // відкрити захисний транзистор Q4
#endif
#if (POWPIN == 1)
 if (BitIsClear(vkr.GlobFlag, TRQ1)) digitalWrite_speed(RELAY220, HIGH); // увімкнути мережу 220В
#endif
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
 lcd.clear();
 return bitRead(vkr.GlobFlag, TRQ1);
}
```

Функція скидання налаштувань

```
// Функція збереження налаштувань
void Saved() {
 EEPROM.put(1, vkr);
 EEPROM.put((int)(sizeof(pam) * vkr.profil + MEM), pam); // Оновити значення пам'яті
}
```

```
const uint8_t data_service[] PROGMEM = {
 POWER, // *Напруга від БП
 POWER_MAX, // *максимально допустима напруга від БП
 INAVOLTKOOF, // коефіцієнт напруги INA226
 (uint8_t) (CURR_MAX * 10), // Максимальний струм зарядника 25.5A
 FAN, // * Тип управління вентилятором
 DEG_ON, // *температура включення вентилятора у градусах
 DEG_OFF, // *температура відключення вентилятора в градусах
 DEG_MAX, // * значення температури для максимальних оборотів вентилятора ШІМ.
 DEG_CUR, // *температура при якій зменшується струм заряду
 (uint8_t)(CURFAN_ON / 100), // * Значення струму в Амперах/100 при якому включається вентилятор. (До 25,5A)
 (uint8_t)(CURFAN_OFF / 100), // * Значення струму в Амперах/100 при якому відключається вентилятор. (До 25,5A)
 (uint8_t) (CURFAN_MAX / 100), // * Значення струму в Амперах / 100 для максимальних оборотів вентилятора ШІМ. (До
25,5A)
 TIME_LIGHT, // *час підсвічування дисплея в хвилинах TIME_LIGHT (до 255 хвилин)
 FREQ_CHARGE, // * Частота силового модуля {0.25, 0.5, 1, 2, 4, 8, 30, 60} кГц
 FREQ_DISCHARGE, // * Частота розрядного модуля {0.25, 0.5, 1, 2, 4, 8, 30, 60} кГц
};

// скидання налаштувань за умовчанням
void Reset_settings(uint8_t sett) {
 lcd.clear();
 printFromPGM((int)(&settings[POINTSETTINGS]));
 int8_t tiks = 0;
 bool ext = true;
 do {
 setCursory();
 lcd.write(RIGHTs); //>
 switch (sett) {
 case 0:
 lcd.print(F(txt5)); // "Exit"
 break;
 case 1:
 lcd.print(F(txt10)); // "All" Стерти все
 break;
 case 2:
 print_mode(3); // "Settings"
 break;
 case 3:
 printFromPGM((int)(&namm9)); // "Calibration"
 break;
 case 4:
 printFromPGM((int)(&nami19)); // "! system param!"
 break;
 }
 ClearStr(12);
 do {
 tiks = myTick();
 } while (! tiks);
 switch (tiks) {
 case ENCLEFT:
 case MINUSCLICK:
 sett = constrain (sett - 1, 0, 4);
 break;
 case ENCRIGHT:
 case PLUSCLICK:
 sett = constrain (sett + 1, 0, 4);
 break;
 case STOPCLICK:
 case ENCHELD:
 return; // вийти
 case ENCCLICK:
 case OKCLICK:
 ext = false; // вийти із циклу
 break;
 }
 } while (ext);
 switch (sett) {
```

```
case 0: break; // вийти
case 1:
//__attribute__ ((fallthrough));
// скинути все
case 2:
// скинути налаштування за замовчуванням
pam.MyFlag = 0b00000000; // - вкл / відкл MyFlag (o,o,o,o,o,precharge,Branimir,Oscillation,Charge) - 0b1000
pam.typeAkb = 0; // - тип акумулятора
pam.Mode = 0; // - Режм заряду
pam.Capacity = 60; // - ємність акумулятора
pam.Voltage = 6; // - кількість осередків акумулятора
pam.Number = 1; // - номер
pam.Round = 1; // - поточне коло
pam.Volt_charge = 14700; // - Напруга заряду - мілівольт * 1000 (макс 65.535 Вольт)
pam.Volt_decrCur = Volt_DCur(); // напруга у якому починається зниження струму заряду
pam.Volt_add_charge = 16200; // - Напруга дозаряду - Мілівольт * 1000 (макс 65.535 Вольт)
pam.Current_charge = 6000; // - Струм заряду - міліампер * 1000 (макс 65.535 Ампер) 0,01C
pam.Current_add_charge = 1200; // - Струм дозаряду - міліампер * 1000 (макс 65.535 Ампер) 0.02C
pam.Current_charge_min = 120; // - Струм завершення заряду - міліампер * 1000 (макс 65.535 Ампер) 0.003C
pam.Current_add_charge_min = 600; // - Струм завершення дозаряду - міліампер * 1000 (макс 65.535 Ампер)
pam.Time_charge_h = 15; // - час заряду, годинник
pam.Time_add_charge_h = 10; // - час дозаряду, годинник
pam.Volt_discharge = 11600; // - Напруга розряду - мілівольт * 1000 (макс 65.535 Вольт)
pam.Current_discharge = 1800; // - Струм розряду - міліампер * 1000 (макс 65.535 Ампер)
pam.Current_discharge_min = 1800; // - Струм завершення розряду - міліампер * 1000 (макс 32,767 Ампер)
pam.Volt_buffer = 13800; // - Напруга буферного (вирівнюючого режиму) - мілівольт * 1000 (макс 65.535 Вольт)
pam.Volt_storage = 13100; // - Напруга зберігання - Мілівольт * 1000 (макс 65.535 Вольт)
pam.Volt_pre_charge = 13800; // - Напруга передзаряду (для сильно розр. Акумул) - Мілівольт * 1000 (макс 65.535 Вольт)
pam.Current_pre_charge = 2000; // - Струм передзаряду (для сильно розр. акумул) - міліампер * 1000 (макс 65.535 Ампер)
pam.Resist_1 = 0;
// - Внутрішній опір акумулятора - МілліОм * 100 - до
pam.Resist_2 = 0; // - Внутрішній опір акумулятора - мілліОм * 100 - після
Res_mem_char();
Res_mem_dischar();

lcd.clear();
lcd.print(F(txt12)); // "Sbros"
for (uint8_t l = 0; l <= PROFIL; l++) {
EEPROM.put((int)(sizeof(pam) * l + MEM), pam); // Оновити значення пам'яті
lcd.write(46); // .....
}
Res_ktc(); // Очистити пам'ять КТЦ
if (sett! = 1) break;
//__attribute__ ((fallthrough));
case 3:
// скинути калібрування за замовчуванням
vkr.GlobFlag = 3; // Q1 – стан силового транзистора, стан INA226/
vkr.Ohms = OHMS; // коофіцмент корекції напруги 0-255
vkr.profil = 0; // поточний профіль 0-6
vkr.Vref = VREF; // Референсна напруга
vkr.shunt = (uint16_t)((float)SHUNT * 100000); // Значення шунта
EEPROM.put(1, vkr);
//if (sett! = 1) break;
//__attribute__ ((fallthrough));
case 4:
// скинути системні параметри за умовчанням
for (byte i = 0; i < sizeof(data_service); i++) {
EEPROM.update(8 + i, pgm_read_byte(&data_service[i]));
}
break;
}
lcd.clear();
if (sett) {
print_mode(3); // "Settings"
lcd.print(F(txt2)); // скидання
setCursory();
lcd.print((sizeof(pam)));
lcd.print(F("B"));
lcd.print((sizeof(vkr)));
```

```
 lcd.write(66); // B
 EEPROM.update(0, MEMVER);
 Delay (3000);
 if (sett == 1 або sett >= 3) resetFunc();
 lcd.clear();
 }
} // скидання налаштувань за умовчанням
```

Налаштування сервісу

```
#if (SERVICE == 1)
 // Зміна сервісних параметрів
void serviceparam(void) {
 lcd.clear();
 int8_t tiks = 0;
 uint8_t arrowPos = 0;
 bool controlState = 0; // клік
 int8_t screenPos = 0; // Номер "екрану"
 int8_t lastScreen = 0; // Попередній номер "екрану"
 bool ext = true;
 do {
 int x = 0; // локальна змінна напрямки
 screenPos = arrowPos / DISPLAYy; // шукаємо номер екрану (0..3 – 0, 4..7 – 1)
 if (lastScreen != screenPos) lcd.clear(); // якщо екран змінився – очищаємо
 lastScreen = screenPos;
 // для всіх рядків
 for (byte i = 0; i < DISPLAYy; i++) {
 lcd.setCursor(0, i); // курсор на початок
 // якщо курсор знаходиться на вибраному рядку
 lcd.write((arrowPos == DISPLAYy * screenPos + i) ? (controlState ? 62 : 126) : 32); // малюємо стрілку або пробіл
 // якщо пункти меню закінчилися, залишаємо цикл for
 if (DISPLAYy * screenPos + i == SETTINGS_AMOUNT) break;
 // виводимо ім'я та значення пункту меню
 printFromPGM((int)(&servicc[DISPLAYy * screenPos + i]));
 lcd.print(F(": "));
 lcd.print(vkr.service[DISPLAYy * screenPos + i]);
 printSimb(); // пробіли для очищення
 }
 // очікування дій користувача
 do {
 tiks = myTick();
 } while (! tiks);

 switch (tiks) {
 case ENCLEFT: / / поворот вліво
 case MINUSCLICK // кнопка мінус
 case MINUSSTEP:
 x = -1;
 break;
 case ENCRIGHT: // Поворот праворуч
 case PLUSCLICK: // кнопка плюс
 case PLUSSTEP:
 x = 1;
 break;
 case ENCCLICK:
 case OKCLICK:
 controlState = !controlState; // рухати курсор / змінювати дані
 break;
 case STOPCLICK:
 case ENCHELD:
 ext = false; // вийти із циклу
 break;
 }
```

```
 if (! controlState) {
 arrowPos = constrain(arrowPos + x, 0, SETTINGS_AMOUNT - 1); // рухаємо курсор // обмежуємо
 } else {
 switch (arrowPos) {
 case FANMODE: // вибір режиму роботи кулера
 vkr.service[FANMODE] = constrain(vkr.service[FANMODE] + x, 0, 2); //Змінюємо параметри
 break;
 case FREQCHARGE: // зміна частоти заряду
 case FREQDISCHAR: // зміна частоти розряду
 vkr.service[arrowPos] = constrain(vkr.service[arrowPos] + x, 0, 7); //Змінюємо параметри частоти
 break;
 default:
 vkr.service[arrowPos] = constrain(vkr.service[arrowPos] + x, 0, 255); //Змінюємо параметри
 break;
 }
 }

 } while (ext);
 lcd.clear();
 print_mode(3); // "Settings"
 lcd.print(F(txt3)); // saved
 lcd.write(63); //?
 if (Choice(30, false)) {
 Saved(); // зберегти налаштування
 resetFunc(); // перезавантажити Ардуїно
 }
}
#endif

// Установка частоти ШІМ на пінах 3 і 11
void Freq(const uint8_t frq) {
 switch (frq) {
 case 0:
 // Піни D3 та D11 - 245 Гц
 TCCR2B = 0b00000101; // x128
 TCCR2A = 0b00000001; // phase correct
 break;
 case 1:
 // Піни D3 та D11 - 490 Гц
 TCCR2B = 0b00000100; // x64
 TCCR2A = 0b00000001; // phase correct
 break;
 case 2:
 // Піни D3 та D11 - 980 Гц
 TCCR2B = 0b00000011; // x32
 TCCR2A = 0b00000001; // phase correct
 break;
 case 3:
 // Піни D3 та D11 - 2 кГц
 TCCR2B = 0b00000011; // x32
 TCCR2A = 0b00000011; // fast pwm
 break;
 case 4:
 // Піни D3 та D11 - 4 кГц
 TCCR2B = 0b00000010; // x8
 TCCR2A = 0b00000001; // phase correct
 break;
 case 5:
 // Піни D3 та D11 - 8 кГц
 TCCR2B = 0b00000010; // x8
 TCCR2A = 0b00000011; // fast pwm
 break;
 case 6:
 // Піни D3 та D11 - 31.4 кГц
 TCCR2B = 0b00000001; // x1
 TCCR2A = 0b00000001; // phase correct
 break;
 case 7:
 // Піни D3 та D11 - 62.5 кГц
```

```
 TCCR2B = 0b00000001; // x1
 TCCR2A = 0b00000011; // fast pwm
 break;
 }
}


 /*
ШІМ на висновках 9 та 10
9 біт, 31250 Гц
TCCR1A = TCCR1A & 0xe0 | 2;
TCCR1B = TCCR1B & 0xe0 | 0x09;

*/
Вивід пунктів налаштувань
// Виведення пунктів налаштувань
void Settings_point(const uint8_t point, const bool edit, const int8_t x) {
 if (! edit) {
 // Відображення налаштувань
 switch (point) {
 case 0:
 PrintVA(pam.Volt_charge, 0, 0, 0, 2); // напруга заряду
 break;
 case 1:
 PrintVA(0, pam.Current_charge, 0, 0, 2); // Струм заряду
 break;
 case 2:
 PrintVA(pam.Volt_decrCur, 0, 0, 0, 2); // Напруга при якому починає знижуватися струм
 break;
 case 3:
 PrintVA(0, pam.Current_charge_min, 0, 0, 2); // Мінімальний струм заряду
 break;
 case 4:
 lcd.print(pam.Time_charge_h); // час заряду
 lcd.write(104); // h
 break;
 case 9:
 PrintVA(pam.Volt_discharge, 0, 0, 0, 2); // напруга розряду
 break;
 case 10:
 PrintVA(0, pam.Current_discharge, 0, 0, 2); // Струм розряду
 break;
 case 11:
 PrintVA(0, pam.Current_discharge_min, 0, 0, 2); // Струм завершення розряду
 break;
 case 12:
 PrintVA(pam.Volt_buffer, 0, 0, 0, 2); // напруга буферний режим
 break;
 case 13:
 PrintVA(pam.Volt_storage, 0, 0, 0, 2); // напруга режиму зберігання
 break;
 case 14: // Передзаряд
 PrintOnOff(bitRead(pam.MyFlag, PRECHAR));
 break;
 case 15:
 PrintVA(pam.Volt_pre_charge, 0, 0, 0, 2); // напруга передзаряду
 break;
 case 16:
 PrintVA(0, pam.Current_pre_charge, 0, 0, 2); // Струм передзаряду
 break;
 case 18:
 lcd.print(pam.Round); //кількість циклів
 break;

 if (pam.typeAkb < 5) {
 //switch (point) {
 case 5:
 PrintVA(pam.Volt_add_charge, 0, 0, 0, 2); // напруга дозаряду
 break;
```

```
case 6:
PrintVA(0, pam.Current_add_charge, 0, 0, 2); // Струм дозаряду
break;
case 7:
PrintVA(0, pam.Current_add_charge_min, 0, 0, 2); // Мінімальний струм дозаряду
break;
case 8:
lcd.print(pam.Time_add_charge_h); // час дозаряду
lcd.write(104); // h
break;
case 17:
PrintOnOff(bitRead(pam.MyFlag, OSCIL)); // режим гойдалки
break;
}
}
} else {
// Зміна налаштувань
switch (point) {
case 0:
// напруга заряду
pam.Volt_charge = constrain (pam.Volt_charge + x * STEP_VOLT, 1000, POWERINT);
PrintVA(pam.Volt_charge, 0, 0, 0, 2);
break;
case 1:
// Струм заряду
pam.Current_charge = constrain (pam.Current_charge + x * STEP_CURR, CUR_CHARGE_MIN, CURRMAXINT);
PrintVA(0, pam.Current_charge, 0, 0, 2);
ChargeTimeCalc(); // Розрахунок часу заряду
break;
case 2:
// Напруга при якому починає знижуватися струм
pam.Volt_decrCur = constrain(pam.Volt_decrCur + x * STEP_VOLT, 1000, pam.Volt_charge);
PrintVA(pam.Volt_decrCur, 0, 0, 0, 2);
break;
case 3:
// Мінімальний струм заряду
pam.Current_charge_min = constrain (pam.Current_charge_min + x * 10, CURMIN, pam.Current_charge);
PrintVA(0, pam.Current_charge_min, 0, 0, 2);
break;
case 4:
// час заряду
pam.Time_charge_h = constrain(pam.Time_charge_h + x, 1, 255);
lcd.print(pam.Time_charge_h);
break;
case 9:
// напруга розряду
pam.Volt_discharge = constrain (pam.Volt_discharge + x * STEP_VOLT, 1000, POWERINT);
PrintVA(pam.Volt_discharge, 0, 0, 0, 2);
break;
case 10:
// Струм розряду
pam.Current_discharge = constrain (pam.Current_discharge + x * STEP_CURR, 50, CURRMAXINT);
PrintVA(0, pam.Current_discharge, 0, 0, 2);
pam.Current_discharge_min = pam.Current_discharge; // Струм завершення розряду
break;
case 11:
// Струм завершення розряду
pam.Current_discharge_min = constrain(pam.Current_discharge_min + x * 10, 50, pam.Current_discharge);
PrintVA(0, pam.Current_discharge_min, 0, 0, 2);
break;
case 12:
// напруга буферного режиму
pam.Volt_buffer = constrain(pam.Volt_buffer + x * STEP_VOLT, 1000, POWERINT);
PrintVA(pam.Volt_buffer, 0, 0, 0, 2);
break;
case 13:
// напруга напруга режиму зберігання
pam.Volt_storage = constrain(pam.Volt_storage + x * STEP_VOLT, 1000, POWERINT);
PrintVA(pam.Volt_storage, 0, 0, 0, 2);
```

```
 break;
 case 14:
 // Передзаряд
 InvBit(pam.MyFlag, PRECHAR);
 PrintOnOff(bitRead(pam.MyFlag, PRECHAR));
 break;
 case 15:
 // напруга передзаряду
 pam.Volt_pre_charge = constrain (pam.Volt_pre_charge + x * STEP_VOLT, 1000, POWERINT);
 PrintVA(pam.Volt_pre_charge, 0, 0, 0, 2);
 break;
 case 16:
 // Струм передзаряду
 pam.Current_pre_charge = constr ain(pam.Current_pre_charge + x * 10, CUR_CHARGE_MIN, CURRMAXINT);
 PrintVA(0, pam.Current_pre_charge, 0, 0, 2);
 break;
 case 18:
 // цикл
 pam.Round = constrain(pam.Round + x, 1, 10);
 lcd.print(pam.Round);
 break;
#if (SERVICE == 1)
 case 19:
 // системні параметри на 3 натискання
#if (ENCBUTT == 0) // 0 - тільки енкодер
 if (enc.clicks == 3) {
 enc.clicks = 0;
#elif (ENCBUTT == 1) // 1 - тільки кнопки
 if (OK.clicks == 3) {
 OK.clicks = 0;
#else // інакше - енкодер + OK + Stop чи все
 if (enc.clicks == 3 або OK.clicks == 3) {
 enc.clicks = 0;
 OK.clicks = 0;
#endif
 serviceparam(); // Зміна системних параметрів
 lcd.clear();
 }
#endif
 break;
 case 20:
 // скидання всіх налаштувань
 Reset_settings(0); // скидання налаштувань за умовчанням
 lcd.clear();
 break;
 //}

 if (pam.typeAkb < 5) {
 //switch (point) {
 case 5:
 // напруга дозаряду
 pam.Volt_add_charge = constrain (pam.Volt_add_charge + x * STEP_VOLT, 1000, POWERINT);
 PrintVA(pam.Volt_add_charge, 0, 0, 0, 2);
 break;
 case 6:
 // Струм дозаряду
 pam.Current_add_charge = constrain (pam.Current_add_charge + x * STEP_CURR, CUR_CHARGE_MIN, CURRMAXINT);
 PrintVA(0, pam.Current_add_charge, 0, 0, 2);
 break;
 case 7:
 // Мінімальний струм дозаряду
 pam.Current_add_charge_min = constrain (pam.Current_add_charge_min + x * 10, CURMIN, pam.Current_add_charge);
 PrintVA(0, pam.Current_add_charge_min, 0, 0, 2);
 break;
 case 8:
 // час дозаряду
 pam.Time_add_charge_h = constrain(pam.Time_add_charge_h + x, 1, 252);
 lcd.print(pam.Time_add_charge_h);
 break;
```

```
 case 17:
 // режим гойдалки
 InvBit(pam.MyFlag, OSCIL); // інвертувати біт
 PrintOnOff(bitRead(pam.MyFlag, OSCIL));
 break;
 }
 }
 ClearStr(10); // Очистити поле
 }
}

// Друк статистики
void Prstatistic(int i, uint32_t time_i, int32_t Ah, int32_t Wh) {
 printFromPGM(i);
 Print_time(time_i, DISPLAYx - 5, 0);
 setCursory();
 PrintVA(0, 0, Ah, Wh, 2);
}
// Висновок статистики
void Statistics_point(uint8_t point) {
 switch (point) {
 case 0:
 // Заряд
 // *Zaryad 24:42*
 // *55.36Ah 366.35Wh*
 Prstatistic((int)(&modeTxt[0]), pam.Time_charge, pam.Ah_charge, pam.Wh_charge);
 break;
 case 1:
 // Дозаряд
 // *Add charge 24:42*
 // *55.36Ah 366.35Wh*
 Prstatistic((int)(&modeTxt[3]), pam.Time_addcharge, pam.Ah_addcharge, pam.Wh_addcharge);
 break;
 case 2:
 // Розряд
 // *Razryad 24:42*
 // *55.36Ah 366.35Wh*
 Prstatistic((int)(&modeTxt[4]), pam.Time_discharge, pam.Ah_discharge, pam.Wh_discharge);
 break;
 case 3:
 // Внутрішній опір
 // *Resist *
 // * 20.3mOm 520A *
 printFromPGM((int)(&menuTxt[6])); // *Resist *
 setCursory();
 lcd.print(((float)pam.Resist_1/100), 1);
 lcd.print(F("mOm "));
 lcd.print(pam.Resist_2); // Висновок пускового струму - напруга ділити на опір
 lcd.write(65); // A
 break;
 default:
 // Висновок значень ємності КТЦ
 {
 int32_t KTC[4] = {0}; // Ah_discharge, Wh_discharge, Ah_charge, Wh_charge,
 uint16_t m = pam.Round; // рахувати кількість циклів у змінну
 m -= (point - 4);
 EEPROM.get(((m << 4) - 16 + MEM_KTC), KTC); // рахувати з пам'яті значення розряду, заряду m-циклу КТЦ
 lcd.print(point - 3); // 1
 lcd.print(F("D:")); // 1D:
 PrintVA(0, 0, KTC[0], 0, 1); // 1D:65.3Ah
 PrintVA(0, 0, 0, KTC[1], 0); // 1D:65.3Ah 255Wh
 setCursory();
 lcd.print(point - 3); // 1
 lcd.print(F("C:")); // 1C:
 PrintVA(0, 0, KTC[2], 0, 1); // 1C:65.3Ah
 PrintVA(0, 0, 0, KTC[3], 0); // 1C:65.3Ah 255Wh
 break;
 }
 }
```

```
}

/*
* *
------------------
*1D:65.3Ah 255Wh *
*1C:66.4AH 257Wh *
Функція захисту
#if (VOLTIN == 1)
// функція захисту від перевищення вхідної напруги
void VinGuard(void) {
 uint16_t volt_in = vin.volt();
 uint8_t vin = 0;
 const uint32_t vmax = (uint32_t)vkr.service[POWERMAX] * 1000; // максимальне напруження від БП
 do {
 if (volt_in > vmax) {
 // якщо напруга перевищила максимальну
 vin++;
 dcdc.Off();
#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, LOW); // відключити живлення реле (вимк. живлення від мережі 220В)
#endif
 Light1(true); // увімкнути підсвічування
 lcd.clear();
 lcd.print(F(txt25)); // "Vin"
 PrintVA(volt_in, 0, 0, 0, 2);
 do {
 Speaker(500); // функція звукового сигналу (час у мілісекундах)
 Delay (10000); // Чекати 10 секунд
 } while (vin > 2);
#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, HIGH); // включити живлення реле (підключення живлення від мережі 220В)
 Speaker(1000); // функція звукового сигналу (час у мілісекундах)
#endif
 } else vin = 0;
 } while (vin);
}
#endif

#if (POWPIN == 1)
// Функція управління реле підключення до мережі 220В
void Relay(void) {
 static uint32_t tvin = 0;
 if (ina.voltsec < 9000) {
 // Якщо напруга підключеної АКБ нижче 9 вольт включається реле 220В.
 digitalWrite_speed(RELAY220, HIGH); // включити живлення реле (підключення живлення від мережі 220В)
 tvin = millis (); // запам'ятати час
 } else if (millis() - tvin > 120000) digitalWrite_speed(RELAY220, LOW); // якщо пройшло 2 хвилини то відключити живлення
реле (вимк. живлення від мережі 220В)
}
#endif

#if (VOLTIN == 1 and POWPIN == 2)
// функція перевіряє напругу від БП і якщо вона більше ніж на Акб включає пін 10 (за замовчуванням), для світлодіода.
void PowerLight(void) {
 if (vin.volt() > ina.voltsec) digitalWrite_speed(RELAY220, HIGH);
 else digitalWrite_speed(RELAY220, LOW);
}
#endif

Функції Z
// функція зберігання акб та буферного режиму
void Storage(bool regim) {
 bool ext = true;
 bool vinoff = true; // прапор відкл БП
 bool vvmin = true; // прапор відключення навантаження
 int8_t tiks = 0;
 uint32_t t_st = 0; // час зберігання
 uint8_t sig = 0; // час відліку сигналу буферного режиму
```

```
 uint16_t vbc = bat.Volt(3) / 100 * 100; // Розрахунок напруги – зниження до 12В
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, HIGH); // увімкнути мережу 220В
#endif
 Freq(vkr.service[FREQCHARGE]); // Встановити частоту роботи силового модуля
 Light1(true); // увімкнути підсвічування дисплея
 {
 int16_t amp = (regim ? (constrain(pam.Capacity << 2, 50, 1000)) : pam.Current_charge); // максимальний струм заряду
 uint16_t vlt = regim? pam.Volt_storage : pam.Volt_buffer;
 PrintVA(vlt, amp, 0, 0, 1); // Висновок напруги та струму заряду
 Delay (3000);
 dcdc.begin(vlt, amp); // задати максимальні значення напруги та струму заряду
 }
 if (!regim) digitalWrite_speed(PIN_13, HIGH); // увімкнути пін 13. Включити реле навантаження
#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
#endif

 disp.start(); // старт відліку часу одна секунда
 ina.start(1); // старт вимірів INA
 dcdc.start();
 while (ext) {
 // цикл роботи
 tiks = myTick(); // опитування кнопок та енкодера
 if (tiks) {
 Light1(true); // Увімкнення підсвічування
 if (tiks == ENCHELD or tiks == STOPHELD) ext = false; // завершити заряд
 }

 if (ina.sample()) {
 if (vinoff) dcdc.Control(); // Регулювання струму та напруги
#if (VOLTIN == 1)
 vin.sample(); // Вимірювання напруги БП
#endif
 }



 // якщо минула секунда
 if (disp.tick()) {
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif

 if (! Regim) Light1 (false); // відключення підсвічування через 3 хвилини у буферному режимі

 Print_time(t_st++, DISPLAYx - 8, 0); // *Storage 12:10:36*

 setCursory(); // *13.10V 6.12A 25C*
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 2); / / Висновок на екран в float - напруга, струм, ампер-годинник, ват-годинник, кільк.
розрядів після коми.
#if (FAN)
 print_tr(kul.fan()); // Читання температури / управління кулером;
#elif (SENSTEMP1)
 print_tr(read_tq1()); // Читання температури
#endif
#if (SENSTEMP2 == 2 and DISPLAYx == 20)
 print_tr(tr_bat.getTempInt()); // Читання температури з датчика 2 акб
#endif
 /*
#if ((DISPLAYx == 20 or DISPLAYx == 16) and DISPLAYy == 4)

#endif
*/

#if (VOLTIN == 1)
```

```
 VinGuard(); // функція захисту від перевищення вхідної напруги
 if (vinoff) {
 if (vin.volt() < ina.voltsec) {
 // якщо зникла напруга від БП то
 dcdc.Off(); // Вимкнути роботу силового модуля
 vinoff = false; // прапор відкл БП
 }
 } else if (vin.volt() > ina.voltsec) vinoff = true; // коли напруга з'явиться увімкнути заряд
#endif

#if (VOLTIN == 1 and POWPIN == 2)
 PowerLight(); // перевіряє напругу від БП і якщо вона більш ніж на Акб включає пін 10 (за замовчуванням), для світлодіода
#endif
 // сигналізація про зниження напруги акб нижче за межу (12V)
 if (ina.voltsec <vbc) {
 if (!sig) {
 Light1(true); // Увімкнення підсвічування
 Speaker(300); // функція звукового сигналу (час у мілісекундах)
 sig = 60; // сигнал раз на 60 секунд
 } else sig--;

 if (! Regim) {
 // Увімкнено буферний режим
 if (vvmin and ina.voltsec < bat.Volt(3)) {
 // якщо навантаження підключено і напруга знизилася менше напруги розряду
 Speaker(300); // функція звукового сигналу (час у мілісекундах)
 vvmin = false;
 digitalWrite_speed(PIN_13, LOW); // вимкнути пін 13. вимкнути реле навантаження
 }
 }
 }
 if (!vvmin and ina.voltsec > pam.Volt_buffer - 200) {
 // коли реле навантаження відключено і напруга збільшилася до максимального то
 Speaker(300); // функція звукового сигналу (час у мілісекундах)
 vvmin = true;
 digitalWrite_speed(PIN_13, HIGH); // увімкнути пін 13. відключити реле навантаження
 }
 }
 // якщо минула секунда
 }
 // цикл роботи
 digitalWrite_speed(PIN_13, LOW); // вимкнути пін 13. вимкнути реле навантаження
 dcdc.Off();
// якщо підключений вентилятор
#if (FAN)
 kul.Off(); // Вимкнути вентилятор
#endif
#if (GUARDA0)
 digitalWrite_speed(PINA0, LOW); // Вимкнути блокування модуля захисту
#endif
}

// функція розрахунку Ач та Втч
int32_t aw_ch(int32_t i) {
 return (i*100/3600);
}

// функція включення блокування модуля захисту залежно від напруги акб
// включає блокування при напруги акб менше 5 Вольт і відключає при напр. більше 6 Вольт
#if (GUARDA0)
void Guard(void) {
 if (ina.voltsec < 5000) digitalWrite_speed(PINA0, HIGH);
 else if (ina.voltsec > 6000) digitalWrite_speed(PINA0, LOW);
}
#endif


// функція скидання значень заряду
void Res_mem_char(void) {
```

```
 pam.Ah_charge = 1; // одиниця для того, щоб Ач і Вт відразу відображалися на дисплеї
 pam.Wh_charge = 1;
 pam.Time_charge = 0;
 pam.Time_addcharge = 0;
 pam.Ah_addcharge = 0;
 pam.Wh_addcharge = 0;
#if (BRANIMIR == 1)
 bitClear(pam.MyFlag, BRANIM);
 bitClear(pam.MyFlag, BRANIMBOIL);
#endif
}

// функція скидання значень розряду
void Res_mem_dischar(void) {
 pam.Ah_discharge = 1; // ємність розряду Ah // одиниця для того, щоб Ач і Вт відразу відображалися на дисплеї
 pam.Wh_discharge = 1; // ємність розряду Вт/год
 pam.Time_discharge = 0; // час розряду
 pam.Ah_addcharge = 0; // Місткість розряду до напруги акумулятора
 pam.Wh_addcharge = 0; // ємність до напруги акумулятора Вт/год
}

// очистити пам'ять значень КТЦ
void Res_ktc(void) {
 EEPROM.put((MEM_KTC - 1), pam.Round); // зберегти кількість циклів
 for (uint16_t i = MEM_KTC; i <= 1019; i++) EEPROM.put(i, 0); // Очистити пам'ять КТЦ
}

// функція очікування (t – мілісекунди)
uint8_t Delay(uint16_t t) {
 uint32_t r = millis();
 int8_t tiks = 0;
 while (millis() - r < t and !tiks) {tiks = myTick(); }
 return tiks;
}

uint16_t Volt_DCur(void) {
 return (pam.Volt_charge + bat.Volt(0)) >> 1;
}



// Функція управління підсвічуванням
void Light1(bool i) {
 if (i) {
 // якщо потрібно ввімкнути підсвічування
 lcd.setBacklight(i); // Увімкнення підсвічування
 light_disp.start(); // старт відліку часу підсвічування дисплей
 } else if (light_disp.tick()) {
 // інакше якщо час вийшов відключення підсвічування
 lcd.setBacklight(i); // відключення підсвічування
 }
}

// Включення розряду і фіксація на певному струмі (amp - міліампер)
void Fixcurrent(int16_t amp) {
 setCursory();
 PrintVA(0, amp, 0, 0, 2);
 float Integ = 0.0;
 uint32_t t = millis();
 ina.start(2); // старт вимірів INA
 do {
 // Збільшення струму розряду
 if (ina.sample()) {
 Integ + = (amp - abs (ina.amperms)) * 0.001f;
 analogWrite_my(PWMDCH, trunc(constrain(Integ, 0, 255)));
 }
 } while (((millis() - t) < 3000) and !myTick());
}
```

```
// калібрування падіння напруги при струмі навантаження, напруги від БП
void Korrect (const uint8_t kof) {
 if (kof and !VOLTIN) return; // якщо обрано калібрування напруги від БП і встановлено дільник напруги на вході вийти з функції
#if (POWPIN == 1)
 digitalWrite_speed(RELAY220, HIGH); // Включити мережу 220В
#endif
 Light1(true); // увімкнути підсвічування
#if (GUARDA0)
 Guard(); // Примусове блокування модуля захисту при напрузі заряду або розряду акб менше 5 Вольт.
#endif
 lcd.clear();
 if (!kof) {
 Freq(vkr.service[FREQDISCHAR]); // Встановити частоту роботи розрядного модуля (4 кГц за замовчуванням)
 lcd.print(F(txt11));
 Fixcurrent(2100); // Встановити струм розряду 1000мА
 lcd.clear();
 } else digitalWrite_speed(RELAY220, HIGH); // включити живлення реле (підключення живлення від мережі 220В)
 bool x = false;
 int8_t i; // + -
 int8_t tiks = 3;
 bool ext = true;
 disp.start(); // старт відліку часу
 ina.start(2); // старт вимірів INA
#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
#endif
 do {
 // Установка напруги
 do {
 // очікування дій користувача
 if (ina.sample()) {
#if (VOLTIN == 1)
 vin.sample(); // Вимірювання напруги БП
#endif
 }
 if (tiks or disp.tick()) {
 // Висновок на дисплей
 switch (kof) {
 case 0:
 lcd.clear();
 PrintVA(ina.voltsec, ina.ampersec, 0, 0, 3);
 setCursory(); // * 12.365V 3.254A *
 Write_x(x); // > or пробіл // *>80 >55 *
 lcd.print(vkr.Ohms);
 lcd.setCursor(8, 1);
 Write_x(!x); // > or пробіл
 lcd.print(((float)vkr.shunt/100));
 lcd.print(F("mOm"));
 break;
#if (VOLTIN == 1) // Вимірювання напруги блока живлення
 case 1:
 setCursorx();
 PrintVA(vin.volt(), 0, 0, 0, 2);
 lcd.write(LEFTs); // <
 lcd.print(vkr.Vref);
 lcd.write(RIGHTs); //>
 break;
#endif
 }
 }
 tiks = myTick();
 } while (! tiks);
 i = 0;
 switch (tiks) {
 case ENCCLICK:
 case OKCLICK:
 x =! X;
 break;
```

```
 case ENCRIGHT:
 case PLUSCLICK:
 i = 1;
 break;
 case ENCLEFT:
 case MINUSCLICK:
 i = -1;
 break;
 case ENCHELD:
 case STOPCLICK:
 ext = false;
 break;
 }
 if (i) {
 switch (kof) {
 case 0:
 if(x) {
 vkr.Ohms = constrain(vkr.Ohms + i, 0, 255);
 } else {
 vkr.shunt = constrain(vkr.shunt + i, 10, 10000); // 1 - 0,1 мОм, 100
00 - 100мОм
 }
 ina.start(1); // старт вимірів INA
 break;
 // Вимірювання напруги блоку живлення
#if (VOLTIN == 1)
 case 1:
 vkr. Vref = constrain (vkr. Vref + i, 1000, 6000);
 vin.begin(vkr.Vref);
 break;
#endif
 }
 }
 } while (ext); // Установка напруги
 digitalWrite_speed(PWMDCH, LOW); // Вимкнути розряд
}

// Функція очікування time_charge – хвилин
void Wait(uint16_t time_charge) {
 lcd.clear();
 lcd.print(F(txt16)); // Pause
 uint8_t tm = 60;
 int8_t tiks = 0;
 disp.start(); // старт відліку часу одна секунда
 ina.start(1); // старт вимірів INA
#if (VOLTIN == 1)
 vin.start(); // старт вимірів напруги від БП
#endif
 do {
 tiks = myTick(); // опитування кнопок
 if (ina.sample()) {
#if (VOLTIN == 1)
 vin.sample(); // Вимірювання напруги БП
#endif
 }
 if (tiks) Light1 (true); // увімкнути підсвічування
 // якщо пройшла секунда виведення на дисплей
 if (disp.tick()) {
 Light1(false); // відключення підсвічування через 3 хвилини
 setCursory();
 PrintVA(ina.voltsec, 0, 0, 0, 2);
 lcd.print(time_charge);
 lcd.print(F(txt20)); // "хвилин"
 tm--;
 // коли минула хвилина
 if (!tm) {
 tm = 60;
 time_charge--;
 }
```

```
#if (VOLTIN == 1)
 VinGuard(); // функція захисту від перевищення вхідної напруги
#endif
#if (POWPIN == 1)
 Relay(); // Функція управління реле підключення до мережі 220В
#endif
#if (VOLTIN == 1 and POWPIN == 2)
 PowerLight(); // перевіряє напругу від БП і якщо вона більш ніж на Акб включає пін 10 (за замовчуванням), для світлодіода
#endif
 }
 } while (time_charge and tiks! = STOPHELD and tiks! = ENCHELD); // цикл у хвилинах або боргове натискання OK
}

// Функція завершення заряду
void End (void) {
 Light1(true); // увімкнути підсвічування
 lcd.clear();
 lcd.print(regim_end); // Висновок причини завершення роботи.
 printSimb(); // пробіл
 print_mode(0); //Виведення режиму
 setCursory();
 int32_t Ah;
 uint32_t tm;
 switch (pam.Mode) {
 case 0:
 // Заряд
 Ah = pam.Ah_charge;
 tm = pam.Time_charge;
 break;
 case 1:
 case 2:
 case 5:
 // Заряд-Дозаряд, Бранімір, КТЦ
 Ah = pam.Ah_charge + pam.Ah_addcharge;
 tm = pam.Time_charge + pam.Time_addcharge;
 break;
 case 3:
 // Дозаряд
 Ah = pam.Ah_addcharge;
 tm = pam.Time_addcharge;
 break;
 case 4:
 // Розряд
 Ah = pam.Ah_discharge;
 tm = pam.Time_discharge;
 break;
 case 6:
 case 7:
 // Зберігання, Буфер.
 Ah = 0;
 tm = 0;
 break;
 }
 PrintVA(0, 0, Ah, 0, 2);
#if (DISPLAYx == 16)
 Print_time(tm, DISPLAYx - 5, 1);
#elif (DISPLAYx == 20)
 Print_time(tm, DISPLAYx - 11, 1);
#endif
#if (DISPLAYy == 4)
 lcd.setCursor(0, 2);
 ClearStr(3);
#endif
 Speaker (2000); // функція звукового сигналу (час у мілісекундах)

 do {
#if (POWPIN == 1)
 ina.sample(); // Читання напруги та струму
```

```
 Relay(); // Функція управління реле підключення до мережі 220В
#endif
 } while (! Delay (10));
 regim_end = 0;
 lcd.clear(); // Очистити дисплей
 Delay (500); // зачекати
}

// Функція вибору Stop/OK. (timer - секунд, час через який повертається false. Якщо timer = 0 то повернення з функції натискання кнопок/енкодера)
bool Choice(uint8_t secund, bool rez) {
 int8_t tiks = 0;
 disp.start();
 bool ext = true;
 do {
 switch (tiks) {
 case ENCLEFT:
 case ENCRIGHT:
 case PLUSCLICK:
 case MINUSCLICK:
 rez = !rez;
 break;
 case ENCCLICK:
 case OKCLICK:
 case STOPCLICK:
 ext = false;
 break;
 }
 lcd.setCursor(4, 1);
 lcd.print((rez ? F(txt17) : F(txt19))); // OK / Stop

 do {
 // цикл очікування дій користувача
 tiks = myTick();
 if (disp.tick()) {
 // якщо пройшла секунда виведення на дисплей
 lcd.setCursor(DISPLAYx - 2, 1);
 lcd.print(secund);
 printSimb();
 secund--;
 }
 } while (!tiks and secund and ext); // цикл очікування дій користувача
 } while (ext and secund); // Очікування натискання або закінчення часу
 return rez;
}

// Функція опитує енкодер і кнопки.
int8_t myTick(void) {
// -1 - left, 1 - right, 16 - click, 2 - hold, 4 - step
#if (ENCBUTT == 0)
 // 0 - тільки енкодер
 if (enc.tick()) {
 if (enc.turn()) return enc.dir(); // -1 - left, 1 - right,
 else {
 int8_t tiks = 0;
 if (enc.click()) tiks = 16;
 if (enc.hold()) tiks = 2;
 return tiks;
 }
 }
#elif (ENCBUTT == 1)
 // 1 - тільки кнопки
 if (OK.tick()) {
 int8_t tiks = 0;
 if (OK.click()) tiks = 26;
 if (OK.hold()) tiks = 12;
 return tiks;
 }
 if (Stop.tick()) {
```

```
 int8_t tiks = 0;
 if (Stop.click()) tiks = 36;
 if (Stop.hold()) tiks = 22;
 return tiks;
 }
 if (Plus.tick()) {
 int8_t tiks = 0;
 if (Plus.click()) tiks = 46;
 if (Plus.step()) tiks = 34;
return tiks;
 }
 if (Minus.tick()) {
 int8_t tiks = 0;
 if (Minus.click()) tiks = 56;
 if (Minus.step()) tiks = 44;
 return tiks;
 }
#elif (ENCBUTT == 2)
 // 2 - енкодер + OK + Stop
 if (enc.tick()) {
 if (enc.turn()) return enc.dir(); // -1 - left, 1 - right,
 else {
 int8_t tiks = 0;
 if (enc.click()) tiks = 16;
 if (enc.hold()) tiks = 2;
 return tiks;
 }
 }
 if (OK.tick()) {
 int8_t tiks = 0;
 if (OK.click()) tiks = 26;
 if (OK.hold()) tiks = 12;
 return tiks;
 }
 if (Stop.tick()) {
 int8_t tiks = 0;
 if (Stop.click()) tiks = 36;
 if (Stop.hold()) tiks = 22;
 return tiks;
 }
#elif (ENCBUTT == 3)
 // 3 - енкодер + усі кнопки
 if (enc.tick()) {
 if (enc.turn()) return enc.dir(); // -1 - left, 1 - right,
 else {
 int8_t tiks = 0;
 if (enc.click()) tiks = 16;
 if (enc.hold()) tiks = 2;
 return tiks;
 }
 }
 if (OK.tick()) {
 int8_t tiks = 0;
 if (OK.click()) tiks = 26;
 if (OK.hold()) tiks = 12;
 return tiks;
 }
 if (Stop.tick()) {
 int8_t tiks = 0;
 if (Stop.click()) tiks = 36;
 if (Stop.hold()) tiks = 22;
 return tiks;
 }
 if (Plus.tick()) {
 int8_t tiks = 0;
 if (Plus.click()) tiks = 46;
 if (Plus.step()) tiks = 34;
 return tiks;
 }
```

```
 if (Minus.tick()) {
 int8_t tiks = 0;
 if (Minus.click()) tiks = 56;
 if (Minus.step()) tiks = 44;
 return tiks;
 }
#endif
 return 0;
}
/*
// ====================== FLAGS Button======================
#define EB_PRESS (1 << 0) 1 // натискання на кнопку
#define EB_HOLD (1 << 1) 2 // кнопка утримана
#define EB_STEP (1 << 2) 4 // імпульсне утримання
#define EB_RELEASE (1 << 3) 8 // кнопка відпущена
#define EB_CLICK (1 << 4) 16 // одиночний клік
#define EB_CLICKS (1 << 5) 32 // сигнал про кілька кліків
#define EB_TURN (1 << 6) 64 // поворот енкодера
#define EB_REL_HOLD (1 << 7) 128 // кнопка відпущена після утримання
#define EB_REL_HOLD_C (1 << 8) 256 // кнопку відпущено після утримання з предв. кліками
#define EB_REL_STEP (1 << 9) 512 // кнопка відпущена після степу
#define EB_REL_STEP_C (1 << 10) 1024 // кнопка відпущена після степу з предв. кліками

// ====================== FLAGS Encoder======================
#define EB_TYPE (1 << 0)
#define EB_REV (1 << 2)
#define EB_FAST (1 << 3)
#define EB_EHLD_M (1 << 4)
#define EB_DIR (1 << 5)
#define EB_ETRN_R (1 << 6)
#define EB_ISR_F (1 << 7)

// 3 - енкодер + усі кнопки
 if (enc.tick()) {
 if (enc.turn()) return enc.dir(); // -1 - left, 1 - right,
 else return sw_action(enc.action()); // 16 - click, 2 - hold, 4 - step
 }
 if (OK.tick()) return sw_action(OK.action()) + 10; // 26 - click, 12 - hold, 14 - step
 if (Stop.tick()) return sw_action(Stop.action()) + 20; // 36 - click, 22 - hold, 24 - step
 if (Plus.tick()) return sw_action(Plus.action()) + 30; // 46 - click, 32 - hold, 34 - step
 if (Minus.tick()) return sw_action(Minus.action()) + 40; // 56 - click, 42 - hold, 44 - step
*/

// функція звукового сигналу (n - час мілісекундах)
void Speaker(uint16_t n) {
// Активний звуковий сигнал
#if (SPEAKER == 2)
 digitalWrite_speed(BUZER, HIGH); // увімкнути сигнал
 Delay(n); // зачекати n - мілісекунд
 digitalWrite_speed(BUZER, LOW); // Вимкнути сигнал

#elif (SPEAKER == 1)
 // пасивний звуковий сигнал
 uint32_t time2 = millis();
 // звуковий сигнал протягом n мілісекунд
 while (millis() - time2 < n and !myTick()) {
 digitalWrite_speed(BUZER, HIGH); // HIGH
 delayMicroseconds(771); // 2314 – частота 432Гц; 1157 – частота 864Гц; 771 – частота 1296Гц
 digitalWrite_speed(BUZER, LOW); // LOW
 delayMicroseconds(771);
 }
#endif
 return;
}

// Розрахунок часу заряду
void ChargeTimeCalc(void) {
 uint8_t v = (uint8_t)((uint32_t)pam.Capacity * 1000 / pam.Current_charge);
 pam.Time_charge_h = constrain(v + ((v + 2 - 1) >> 1), 1, 100); // Розрахунок часу заряду
```

```
}

// Обчислення відсотка заряду (максим напруження заряду, поточна напруга заряду, поточний струм заряду, мінімальний струм заряду)
uint8_t Percentage(uint16_t volt_max, uint16_t volt, int16_t current, int16_t curr_min) {
 uint16_t vmin = bat.Volt(3); // Мінім напруга (напруга розряду)
 if (volt < vmin) return 0; // якщо поточна напруга акб менше напруги розряду, то повернути 0%
 uint8_t vperc = map(volt, vmin, volt_max, 1, 50); // Відсоток заряду за напругою 50%
 vperc + = (uint8_t) (curr_min * 50 / current); // Додаємо відсоток заряду по струму 50%
 return constrain(vperc, 1, 100);
}

// дуже хитра функція для друку з PROGMEM
void printFromPGM(int charMap) {
 uint16_t ptr = pgm_read_word(charMap); // Отримуємо адресу з таблиці посилань
 while (pgm_read_byte(ptr) != NULL) {
 // весь рядок до нульового символу
 lcd.print(char(pgm_read_byte(ptr))); // Виводимо в монітор чи куди нам треба
 ptr++;
// наступний символ
 }
}

// Завантаження в пам'ять дисплея своїх символів (стан акб)
void lcdmysimbol(void) {
 for (uint8_t x = 0; x < 8; x++) {
 lcd.createChar(x, &simb_array[x][0]); // завантажуємо символи на згадку дисплея
 }
}


#if (SENSTEMP2 == 2)
// контроль температури Акб з датчика NTC підключеного до пін А6 Ардуїно
bool Control_trAkb() {
 bool tr = true; // Повернути якщо температура акб у нормі
 int16_t tr_akb = tr_bat.getTempInt(); // Читання температури з датчика 2 акб
 int16_t curr_charge_init = dcdc.getCur_charge(); // Струм заряду
 // якщо акб свинцево-кислотний, то проводиться коригування напруги заряду 30 мВ на кожен градус Цельсія при відмінності температури від +25°C.
 if (pam.typeAkb <= 4) dcdc.setVolt_charge(dcdc.getVolt_charge() - (tr_akb - 25) * 30); // Pb
 else if (pam.typeAkb <= 6 або pam.typeAkb == 8) { // "Li-ion" "LiFePo4" "NiCd/Mh"
 // якщо температура акб менше 10 або більше 40 гр, то відключити заряд
 if (tr_akb < 10 або tr_akb > 40) {
 tr = false;
 dcdc.Off();
 } else if (tr_akb > 35) dcdc.setCur_charge(curr_charge_init - ((tr_akb - 35) * 10 * curr_charge_init / 100)); // якщо темпратура акб більше 35 гр, то зменшувати струм заряду на 10% на градус.
 else if (tr_akb < 15) dcdc.setCur_charge(curr_charge_init - ((5 - (tr_akb - 10)) * 10 * curr_charge_init / 100)); // якщо темпратура акб менше 15 г то зменшувати струм заряду на 10% на градус.
 else tr = true; // t_akb = true; // інакше продовжувати заряд
 } else if (pam.typeAkb == 7 and tr_akb > 40) dcdc.setCur_charge(curr_charge_init - ((tr_akb - 40) * 10 * curr_charge_init / 100)); // "LiTit" // якщо температура акб більше 40 гр то зменшувати струм заряду на 10% на градус.
 return tr;
}
#endif

// функція відправлення значень до мережного порту
// час, напруга, струм, А/год, Вт/год, темп Акб, темп Q1, напруга БП
#if (LOGGER == 1)
void Serial_out(float volt, float amperSred, uint8_t tQ1, float volt_in, float Achas, uint8_t tAkb) { //
 // const char * simb = ";";
 Serial.print(volt/1000, 3); //1 напруга на АКБ (мілівольт)/1000_____________________
 Serial.print(";"); // символ між значеннями_____________________________________
 Serial.print(amperSred/1000, 3); //2 струм АКБ (міліампер)/1000_____________________________
 Serial.print(";"); // Символ між значеннями_____________________________________
 Serial.print(tQ1, 1); //3 Температура на Q1 (градуси)
 Serial.print(";"); // Символ між значеннями_____________________________________
 Serial.print(volt_in/1000, 3); //4 напруга від БП (мілівольт)/1000_____________________ vin
 Serial.print(";"); // Символ між значеннями_____________________________________
```

```
 Serial.print(Achas/100000, 3); //5 Отримана_віддана ємність АКБ (А/год)) /100000_____
 Serial.print(";"); // символ між значеннями_____
 Serial.print(1); //6 Температура АКБ (градуси)
 //Serial.print(";");
 //Serial.print(7); //7 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері. потрібна корекція скетчу WiFi перехідника
 //Serial.print(";");
 //Serial.print(8); //8 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(9); //9 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(10); //10 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(11); //11 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(12); //12 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(13); //13 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(14); //14 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(15); //15 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(16); //16 підключений: значення на головній сторінці WiFi перехідника і на логері
, вимкнено: значення лише на логері.
 //Serial.print(";");
 //Serial.print(17); //17 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 Serial.println(";"); // Знак перенесення рядка.
 }
#endif

// час, напруга, струм
#if (LOGGER == 2)
void Serial_out(float volt, float amperSred, uint8_t tQ1, float Achas) {
 // const char * simb = ";";
 Serial.print(volt/1000, 3); //1 напруга на АКБ (мілівольт)/1000_____
 Serial.print(";"); // символ між значеннями_____
 Serial.print(amperSred/1000, 3); //2 струм АКБ (міліампер)/1000_____
 Serial.print(";"); // символ між значеннями_____
 Serial.print(tQ1, 1); //3 Температура на Q1 (градуси)
 Serial.print(";"); // символ між значеннями_____
 Serial.print(1); //4 напруга від БП (мілівольт)/1000_____
 Serial.print(";"); // символ між значеннями_____
 Serial.print(Achas/100000, 3); //5 Отримана_віддана ємність АКБ (А/год)) /100000_____
 Serial.print(";"); // символ між значеннями_____
 Serial.print(1); //6 Температура АКБ (градуси)
 //Serial.print(";");
 //Serial.print(7); //7 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері. потрібна корекція скетчу WiFi перехідника
 //Serial.print(";");
 //Serial.print(8); //8 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(9); //9 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(10); //10 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
```

```
 //Serial.print(11); //11 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(12); //12 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(13); //13 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(14); //14 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(15); //15 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(16); //16 підключений: значення на головній сторінці WiFi перехідника і на логері, відключений: значення тільки на логері.
 //Serial.print(";");
 //Serial.print(17); //17 підключений: значення на головній сторінці WiFi перехідника та на логері, відключений: значення тільки на логері.
 Serial.println(";"); // Знак перенесення рядка.00
}
#endif
```