МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
"_____"_____ 2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Система дослідження реологічних властивостей індустріальних масел

Виконавець: _____Харьков М.В.

Керівник: _____ Нечипорук В.В.

Нормоконтролер: _____Тупота Є.В.

Київ 2020

РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему «Система дослідження реологічних властивостей індустріальних масел»: 81 с., 22 рис., 3 табл., 22 літературних джерела, 2 додатки.

АВТОМАТИЗОВАНІ СИСТЕМИ УПРАВЛІННЯ, СИСТЕМИ ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ, МАШИНА ТЕРТЯ, *LPC1768, ARM*

Об’єкт дослідження – автоматизоване керування складними об’єктами.

Предмет дослідження – апаратно-програмна система керування машиною тертя.

Мета дипломного проекту – розробити апаратно-програмно систему керування машиною тертя для виконання збору, обробки, збереження інформації про реологічні властивості зразків індустріальних масел.

Методи проектування – теоретичне ознайомлення з існуючими аналогами систем керування, використання здобутих знань для розробки програмно-апаратної системи керування. Використання *Visual Studio, mbed IDE*.

Прогнозні припущення щодо розвитку об’єкта дослідження – можливість розширення периферійної складової шляхом додавання нових пристроїв та підключенням їх, програмної складової – інтеграція нового функціоналу (підмодуль обробки отриманої інформації), збільшення кількості команд у протоколі комунікації інтерфейсу управління та *LPC1768*, заміни технології передачі на бездротову, реалізація віддаленого доступу за допомогою веб-технологій. Посилення аналітичної складової програмної частини системи.

Результати дипломної роботи можна розділити на дві групи: теоретичні та практичні. Теоретичні результати можна використовувати при розробці нових автоматизованих систем управління, практичні – при експериментальних дослідженнях та діагностиці вузлів тертя при терті поверхонь із композиційних матеріалів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

АЕ – акустична емісія

АОП – апаратно-обчислювальна платформа

ПТ – пара тертя

CLI – *common intermediate language* (специфікація загальномовної структури)

ВСТУП

Найбільш важливою складовою частиною наукових досліджень є експерименти. Це один з основних способів отримання нових наукових знань. Більше двох третин трудових ресурсів науки витрачається на проведення експериментів. В основі експериментального дослідження лежить експеримент, що являє собою науково поставлене дослідження або спостереження за явищем в точно поставлених умовах, що дозволяють слідкувати за його протіканням, керувати ним, відтворювати його кожен раз при повторенні цих умов. Саме поняття експеримент означає дію (або комплекс дій), направлену на створення певних умов в цілях відтворення того чи іншого явища і по можливості найбільш чистого, тобто не спотвореного іншими явищами.

Експеримент, як правило, має на меті уточнити характеристики обладнання, явища, процесу чи реального об’єкта, визначити надійність його роботи в непередбачуваних або аварійних режимах, перевірити правильність теоретичних розрахунків, тощо. При цьому враховується весь обсяг факторів, що діють на досліджуваний об’єкт.

Лабораторні досліди приводять з використанням типових приборів, спеціальних моделюючих установок, стендів, обладнання. Ці досліди дозволяють найбільш повно і доброякісно вивчити вплив одних характеристик на інших, причини виникнення явищ, тощо, за допомогою ізоляції досліджуваного середовища від другорядних факторів, що затінюють його перебіг. Проведення дослідів на такому устаткуванні дозволяє багатократно відтворювати хід досліджуваного процесу в строго фіксованих умовах, що під час проведення натурного експерименту здебільшого зробити дуже важко.

Лабораторні досліди в випадку достатньо повного наукового доведення експерименту дозволяють отримати цінну наукову інформацію з мінімальними затратами на їх проведення.

Підвищення ефективності фундаментальних і прикладних наукових досліджень стає важливим фактором прискорення науково-технічного прогресу. Особливе

значення для підвищення ефективності науки набуває автоматизація наукових досліджень, що дозволяє отримувати більш точні та повні моделі досліджуваних об'єктів і явищ, прискорювати хід наукових досліджень і знижувати їх трудомісткість, вивчати складні об'єкти і процеси, дослідження яких традиційними методами важко або неможливо.

Проведення експериментальних досліджень на спеціальному обладнанні для триботехнічних випробувань потребує постійного контролю і реєстрації багатьох параметрів, а за необхідності і внесення коригуючих змін в план проведення випробувань. Такими параметрами є: температура зразків та робочої рідини, момент тертя, навантаження, частота та кількість обертів зразків.

Для автоматизації складних систем застосовують автоматизовані системи управління.

Автоматизовані системи управління поділяються на два основних типи:

– автоматизовані системи управління технологічними процесами (АСУТП);

– автоматизовані системи організаційно-економічного управління підприємством.

На жаль, обладнання триботехнічних лабораторій, яке використовується в наукових дослідженнях, а особливо матеріальна база електронних систем забезпечення автоматичного контролю необхідних параметрів є морально застарілою. Модернізація такого обладнання зводиться до використання сучасної елементної бази в системах автоматизованого управління технологічними процесами та контролю параметрів в трибологічних дослідженнях. Але значна вартість подібних систем та закриті алгоритми їх функціонування не дозволяють в повній мірі провести модернізацію систем управління та вимірювання для вирішення проблем, що висуваються до даного обладнання. Тож використання засобів автоматизації на сучасній елементній базі, що відповідає вимогам – швидше, точніше, дешевше є актуальною задачею при проведенні триботехнічних випробувань.

Сучасна елементна база дозволяє створювати інформаційно-вимірювальні та інформаційно-управляючі системи, використовуючи в своєму складі не тільки аналогові, але й цифрові первинні перетворювачі. Це значно розширює можливості

систем контролю, збору та обробки інформації. А також дає можливість реалізувати системи автоматизованого управління з використанням інтелектуальних вимірювальних систем і віртуальних приладів [1].

Застосування сучасної елементної бази в системах автоматизації електромеханічних систем, дозволяє перейти на більш високий рівень вирішення широкого спектра завдань з одночасним поліпшенням швидкісних і енергетичних показників процесу.

Метою даної роботи є створення автоматизованої апаратно-програмної системи керування, що має виконувати завдання збору даних датчиків, що знімають показники апаратури та середовища проведення дослідження та зберігання цих результатів на носій інформації персонального комп'ютера, завдання перетворення команд оператора в контролюючі сигнали апаратури дослідження при випробуваннях моделі зубчатої передачі в умовах змащування на машині тертя СМЦ-2.

Об'єктом дослідження виступає автоматизоване керування складними об'єктами.

Предметом дослідження є апаратно-програмна система керування машиною тертя.

Практична цінність результатів роботи полягає у наданні можливості:

– відновити проведення триботехнічних випробувань, що дозволить отримувати нові наукові знання, що можуть бути застосовані для вирішення проблем тертя та змащення у різноманітних механічних системах;

– автоматизувати проведення дослідних випробувань, що дозволить виконувати довготривалі експериментальні дослідження, підвищить їх точність та якість, візьме на себе рутинні обов'язки оператора такого обладнання;

– модернізувати елементну базу, що дозволить зменшити вартість, габарити, затрати на енергоживлення та спростить схему керуючого та знімаючого показники обладнання;

– дозволить розширити АСУ інтеграцією підмодулів, а саме підмодулем обробки та аналізу отриманої інформації;

– відкриє можливість до модернізації або використанні додаткового до існуючого обладнання з мінімальними трудозатратами, що дозволить проводити дослідження на тертя і зношення розширюючи їх шляхом використання більш сучасних і точних методик.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Принципи функціонування автоматизованих систем керування

Система управління – систематизований (строго певний) набір засобів збору відомостей про підконтрольний об'єкт і засобів впливу на його поведінку, призначений для досягнення певних цілей. Об'єкт системи управління може складатися з інших об'єктів, які можуть мати постійну структуру взаємозв'язків.

Технічна структура управління – пристрій або набір пристроїв для маніпулювання поведінкою інших пристроїв або систем.

Структури управління поділяють на два великі класи:

– автоматизована система управління (АСУ) – з участю людини в контурі управління;

– система автоматичного управління (САУ) – без участі людини в контурі управління [2].

Об'єктом управління може бути будь-яка динамічна система або її модель. Стан об'єкта характеризується деякими кількісними величинами, що змінюються в часі, тобто змінними стану. Для технічних об'єктів це механічні переміщення (кутові або лінійні) і їх швидкість, електричні змінні, температури і т.д.. Аналіз і синтез систем управління відбувається за допомогою засобів спеціального розділу математики – теорії управління [3].

Система управління контролює, керує, регулює поведінку інших пристроїв або систем з використанням контуру управління. Існує два загальних класи контурів керування: відкритий і замкнутий. У системі управління з відкритим контуром керуюча дія контролеру не залежить від змінної процесу. Прикладом цього є котел центрального опалення, керований тільки таймером. Керуючий вплив – це увімкнення або вимкнення котла. Ця технологічна змінна є температурою будівлі. Цей регулятор керує системою опалення постійно, незалежно від температури

будинку. У замкнутій системі управління, керуюча дія контролеру залежить від бажаної і фактичної змінної процесу. У випадку аналогії котла, використовується термостат для контролю температури будівлі оператором. Контролер за допомогою механізму зворотного зв'язку постійно отримує значення температури будівлі і залежно від отриманого значення регулює температуру, щоб фактичне значення було близьким або рівним до бажаного.

Зворотній зв'язок – це властивість системи з замкнутим контуром, яка дозволяє порівнювати вихід (або іншу керовану змінну) з входом в систему (або входом до деякого іншого внутрішньо розташованого компонента або підсистеми), щоб була сформована відповідна керуюча, як деяка функція виведення і введення [2]. Зворотній зв’язок створює можливості для автоматизації існуючої системи.

Для покращення роботи системи може бути створений комбінований контур керування – система, в якій вхідними даними для прийняття рішень контролером є як зовнішні (задані і збудникові), так і внутрішні (контрольні) сигнали.

Автоматизація управління – це якісно новий рівень вирішення всіх завдань управління, перехід до якого можливий і ефективний при дотриманні ряду принципів, що визначилися практикою створення та експлуатації автоматизованих систем управління.

Автоматизована система управління – комплекс апаратних і програмних засобів, а також персоналу, призначений для управління процесами в рамках технологічного процесу [4].

Автоматизація дозволяє частково або повністю звільнити людину від виконання циклічних процесів, або процесів, що виконуються по строго заданому алгоритму. Поняття «автоматизований», на відміну від «автоматичний», підкреслює необхідність участі людини в окремих операціях, з метою збереження контролю над процесом, так і з метою зі складністю або недоцільністю автоматизації окремих операцій. В даний час важко собі уявити виробництво, де все або частину процесів контролюються без відома людини, повідомляючи її тільки в разі несправності або передаварійної ситуації.

На відміну від локальних систем автоматики, автоматизована система управління має контроль над усіма вузлами обладнання, відповідно може більш точно запобігати розвитку аварійних ситуацій.

Система автоматичного керування підтримує або покращує функціонування керованого об'єкта. У ряді випадків допоміжні операції для АСУ – пуск, зупинка, моніторинг, налаштування тощо - також можуть бути автоматизованими.

Новий етап автоматичного управління характеризується впровадженням електронних елементів і пристроїв для автоматизації та дистанційного керування. Електронні досягнення були відповідальними за появу високоточних систем стеження та наведення, дистанційного керування та телеметрії, а також систем автоматичного моніторингу та корекції [5].

Прилади контролю та автоматичного регулювання класифікують наступним чином:

– вимірювальні перетворювачі (датчики), що дозволяють виміряти контрольовану величину і перетворити її в сигнал, зручний для подальшого використання;

– вторинні прилади, що діють спільно з датчиками і дозволяють записувати, показувати контрольовану величину, сигналізувати про її будь-які відхилення, перетворювати і передавати сигнал на автоматичні регулятори;

– автоматичні регулятори, які сприймають сигнал від датчика про дійсне значення регульованої величини, а від задавача – про задане значення регульованої величини, а також оцінюють відхилення фактичного від бажаного значення та впливають через виконавчі пристрої на регулюючий орган для приведення регульованої величини у відповідність із заданим значенням;

– виконавчі пристрої, що дозволяють впливати на регулюючий орган, для зміни подачі ресурсів в регульований об'єкт;сигнальні пристрої, що дозволяють сповістити про досягнення регульованою величиною певного рівня або аварійного значення.

З появою однокристальних мікро-ЕОМ пов’язують початок ери масового застосування комп’ютерної автоматизації в області управління. Ця обставина закріпила термін «контролер». Мікроконтролер – мікросхема, призначена для

управління електронними пристроями. Типовий мікроконтролер поєднує на одному кристалі функції процесора і периферійних пристроїв, має ОЗУ та/або ПЗУ.

Застосування найостанніших розробок в області мікропроцесорної техніки дозволяє об'єднувати системи автоматизації на автоматизовані системи управління технологічними процесами вищого рівня, крім збору та діагностики інформації, такі системи займаються груповим регулюванням роботи агрегатів та інших технологічних схем.

Програмовані логічні контролери досягли колосальної продуктивності, що в комплексі з резервуванням робить їх роботу максимально безвідмовною і швидкодіючою.

Принцип роботи всіх АСУ, застосовуваних для створення автоматичних комплексів, в сукупності дуже схожий. З датчиків (вимірювачів), зчитуючих показання поточних параметрів або характеристик, сигнал надходить на контролер і там оброблюється в відповідності до закладеної програми. Оброблений сигнал передається на комп’ютер, який відображає і зберігає зібрану інформацію. В той самий час з контролера керуючі сигнали видаються на виконуючі механізми для підтримання заданих параметрів. Всіма процесами і діями керує контролер, що значно підвищує надійність АСУ на відміну від залучення людського фактору до рутинних операцій.

Автоматизовані системи керування будуються у відповідності до принципів модульності. Такий підхід дозволяє проектувати не тільки окремі АСУ будь-якої конфігурації, але і об’єднувати їх в єдиний комплекс, що забезпечує обмін інформацією і взаємодію між ними. Вся зібрана інформація зберігається на основному ПК і оператор має змогу переглядати, обробляти та аналізувати дані.

Головною задачею автоматичних систем керування технологічними процесами є підвищення ефективності виробництва і якості продукції. Успішно реалізована система керування технологічним процесом дозволяє нарощувати виробничі потужності, збільшувати ефективність виробництва і колосально збільшує рівень якості. Також за допомогою АСУ відбувається автоматизація праці, що позитивно відзначається на здоров’ї персоналу.

Усі ці позитивні сторони застосування АСУТП виробництва також проектуються на процеси які вивчаються за допомогою експериментальних наукових досліджень. З використанням автоматизованих систем керування збільшується правильність проведення досліду оскільки для експерименту критично важливо підтвердити виникнення певного явища під час багаточисельних випробувань з ідентичними налаштуваннями системи та зі зменшенням (або виключенням) ймовірності впливу небажаних зовнішніх факторів.

1.2. Опис реологічних властивостей індустріальних масел

Реологічні вимірювання надають значення показників властивостей матеріалу, а фундаментальні реологічні концепції передбачають, яким чином розуміти результати цих випробувань і як їх використовувати для вирішення прикладних задач. Те, яким чином отримані значення реологічних характеристк матеріалу можна застосувати для вирішення конкретних технологічних або інших прикладних проблем, залежить від глибини розуміння фізичних явищ, що лежать в основі цих проблем. В деяких випадках реологічні властивості матеріалу безпосередньо визначають його поведінку при використанні, в інших – їх розглядають разом з іншими властивостями речовини.

Реологія – метод визначення фізичних властивостей матеріалу, які залежать від його структури. Оскільки реологічні методи дають однозначні фізично значимі кількісні оцінки властивостей речовини, вони можуть бути зіставлені зі структурою, хімічними або фізичними параметрами матеріалу. Таким чином реологічні властивості корелюють зі структурою матеріалу і можуть використовуватись для її характеристики.

Роль мастил відіграють дуже важливу роль у сфері трибології. Змащення полягає в тому, щоб згладити рух однієї поверхні над іншою та підтримувати в’язкопружну поведінку. Змазки зазвичай використовуються для змащування та зменшення тертя, а отже і зносу, поверхонь, що труться, та ефективного теплообміну завдяки хорошій теплопровідності. Більшість мастильних матеріалів – це рідини

(наприклад мінеральні та синтетичні оливи, кремнієві рідини, вода тощо). Підбір мастильного матеріалу дуже важливий для забезпечення довшого терміну експлуатації механізму тертя.

Для вибору відповідної мастильної речовини необхідно знати її властивості, систему змащення застосовуваних машин, умови роботи машин, вартість оливи. Загальними властивостями мастила є: в'язкість, індекс в'язкості, щільність, стисливість, поверхневий натяг, температура помутніння, температура застигання, температура спалаху, коефіцієнт тертя, висока температура кипіння, низька температура замерзання, термостійкість, запобігання корозії, висока стійкість до окислення тощо. Найважливішою властивістю є в'язкість. В'язкість – це функція температури та тиску. Взаємозв'язок між в'язкістю і температурою, а також взаємозв'язок між в'язкістю і тиском також важливі для реології мастила, а також для терміну служби елементів машини. Подібно до того, як підвищення температури знижує в'язкість масла, також збільшення тиску викликає підвищення його в'язкості. В останні роки залежність в'язкості від тиску стала важливим параметром масла, щоб зрозуміти його ефективність, особливо в умовах високих температур, тому вимірювання в'язкості стає важливим інструментом цього. Реологія - це дослідження потоку речовини: переважно рідин, але також м'яких твердих речовин або твердих речовин в умовах, в яких вони течуть, а не деформуються.

Важко уявити будь-який тип машинних робіт які виконуються без змащення. Одним з найбільших застосувань мастильних матеріалів, у вигляді моторного масла, є захист двигунів внутрішнього згоряння в моторних транспортних засобах та обладнанні, що працює на двигуні. Зазвичай мастильні матеріали містять 90% базового масла (найчастіше нафтові фракції – мінеральні масла) і менше 10% добавок [6].

Технічні масла, що використовуються в промисловості мають задовольняти певним вимогам, серед яких є експлуатаційні, фізичні (теплові), екологічні і реологічні. Реологічні властивості пов'язані з в'язкістю, в'язкістно-температурними, релаксаційними, адгезійними і іншими характеристиками олив. Цим може бути

пояснений великий інтерес при виборі для експлуатації мастил, що мають гарні параметри за цими властивостями.

Моторне масло в двигуні служить для зниження температури, тертя і зносу деталей двигуна, що взаємодіють між собою за рахунок створення на їх поверхнях міцної масляної плівки. Одночасно моторні масла повинні забезпечити:

– ущільнення зазорів в сполученнях працюючого двигуна;

– ефективне відведення тепла від деталей, видалення з зон тертя продуктів зносу;

– надійний захист робочих поверхонь деталей двигуна від корозійного впливу продуктів окислення масла і згоряння палива;

– запобігання утворенню всіх видів відкладень (нагар, лаки, зольні відкладення) на деталях двигуна при його роботі на різних режимах;

– збереження початкових властивостей як в різноманітних умовах застосування, так і при тривалому зберіганні;

– малу витрату масла при роботі двигуна;

– великий термін служби масла до його заміни без шкоди для надійності двигуна.

Виконання зазначених функцій моторними маслами можливо тільки в тому випадку, якщо їх якість буде задовольняти перерахованим експлуатаційним вимогам:

– висока змиваюча та очисна властивості, що забезпечує чистоту деталей двигуна;

– висока термічна і термоокислювальна стабільність, що дозволяє підвищити граничну температуру нагріву масла в картері і збільшити термін заміни;

– достатні протизносні властивості, що забезпечуються міцністю масляної плівки, потрібної в'язкісті при високій температурі і високому градієнті швидкості зсуву, здатністю хімічно модифікувати поверхню металу при граничному терті, нейтралізувати кислоти, які утворюються при окисленні масла і продуктів згоряння палива;

– надійний захист поверхонь, що труться і інших металевих деталей від корозійного впливу як під час роботи, так і при зберіганні автомобілів;

– стійкість до старіння, здатність протистояти зовнішнім впливам з мінімальним погіршенням властивостей;

– пологість в'язкісно-температурної характеристики, забезпечення холодного пуску, прокачуваності при ньому і надійного змазування в екстремальних умовах при високих навантаженнях і температурі навколишнього середовища;

– висока стабільність при транспортуванні і зберіганні в регламентованих умовах;

– мала випаровуваність, низька витрата на чад [7].

Експлуатація транспортних засобів, машин і механізмів невідворотно пов’язана з змінами якісних і кількісних показників застосовуваних в них масел. Закладений в процесі виробництва потенційний ресурс мастильних матеріалів витрачається в процесі експлуатації. Інтенсивність цього процесу пов’язана з експлуатаційними факторами і ступенем їх впливу на процес старіння мастил.

Ресурс мастильних матеріалів визначається якісними показниками мастил; технічним станом механічних систем, в яких воно застосовується; навантажувальними і швидкісними режимами роботи; температурними режимами; ступенем впливу зовнішнього середовища на процес експлуатації техніки і продуктивністю системи фільтрації. Якісні показники масел мастил залежать від базової основи і комплекту домішок. Базові моторні і трансмісійні масла поділяються на мінеральні, частково синтетичні і синтетичні, в комплекти домішок забезпечують належні параметри експлуатаційної якості.

1.3. Процес проведення випробувань на тертя та зношення

Експлуатація транспортних засобів, машин і механізмів неминуче пов'язана зі змінами якісних і кількісних показників застосовуваних в них мастил. Закладений в процесі виробництва потенційний ресурс мастильних матеріалів витрачається в процесі експлуатації. Інтенсивність цього процесу пов'язана з експлуатаційними чинниками та рівнем їхньої впливу на процес старіння мастил.

Ресурс мастильних матеріалів визначається якісними показниками товарних мастил; технічним станом механічних систем, в яких воно застосовується; навантажувальні і швидкісними режимами роботи; температурними режимами; ступенем впливу навколишнього середовища на процес експлуатації техніки і продуктивністю системи фільтрації. Якісні показники товарних мастил залежать від базової основи і комплекту присадок. Базові моторні та трансмісійні масла підрозділяють на мінеральні, частково синтетичні і синтетичні, також вводяться комплекти присадок забезпечують необхідні параметри експлуатаційної якості.

При експлуатації вузлів тертя змінюються як геометричні розміри так і структура, фізико-механічні властивості, а також напружено-деформований стан поверхонь фрикційного контакту. Тому для забезпечення експлуатаційної надійності вузлів тертя необхідно проводити контроль їх технічного стану. Це особливо актуально, в зв'язку з тим, що ефект від експлуатації за технічним станом еквівалентний вартості близько 30 відсотків загального парку машин [8].

Аналіз науково-технічної літератури показує, що основною метою дослідження стану поверхонь фрикційного контакту є: вирішення проблеми підвищеного зношування поверхонь фрикційного контакту; запобігання виникненню зношуваності для забезпечення необхідної працездатності вузлів тертя; визначення типу та марки матеріалів, які використовуються в фрикційному з’єднанні; визначення виду зміцнюючої обробки фрикційних матеріалів; визначення механізмів зношування поверхонь фрикційного контакту.

Діагностика та контроль стану вузлів тертя може включати в себе аналіз багатьох процесів та вимірювання різноманітних фізичних величин. Такі вимірювання спрямовані, в першу чергу, на дослідженні механізмів зношування та руйнування поверхонь, а також способів їх оцінки.

Методи діагностики та контролю вузлів тертя також розділяють на статичні та динамічні. Статичні методи, основані на реєстрації деяких параметрів після дії контактно-фрикційної взаємодії спряжених поверхонь. Група даних методів характерна для лабораторних досліджень та випробувань. Одним з найбільш поширених статичних методів є фрактографічний аналіз. При вивченні рельєфу

поверхонь та структурних характеристик поверхневого шару матеріалів вузлів тертя використовується оптична й електронна мікроскопія.

Динамічні пасивні методи контролю та діагностики стану поверхонь вузлів тертя основані на фізичних явищах (теплові, електричні, акустичні, реологічні та ін.) та дозволяють отримувати інформацію про процеси поверхневого руйнування матеріалів. До основних пасивних методів відносять методи вібродіагностики та термодіагностики, шумову діагностику, акустичну емісію та ін.

Можливість отримання достовірної інформації про процеси пружно-пластичної деформації та руйнування поверхонь матеріалів зумовило до широкого використання методу акустичної емісії (АЕ) в дослідженні, контролі та діагностиці стану вузлів тертя [9].

Експрес-оцінка антифрикційних і протизносних властивостей рідких мастильних композицій і пластичних мастильних матеріалів на машинах тертя є перспективним напрямком удосконалення методів лабораторних випробувань.

Використовують різні схеми для реалізації таких випробувань на машинах тертя. За геометрією контакту поверхонь тертя ці схеми можуть бути розділені на дві групи: з постійною і змінною площею контакту. Площа контакту в першому випадку визначається геометричними розмірами поверхонь тертя. Друга група включає в себе випробування матеріалів з початковим точковим або лінійним контактами. За схемою тертя з початковим точковим контактом працює чотирьохкулькова машина тертя, а початковим лінійним контактом – машина тертя СМЦ-2 при використанні циліндричних зразків.

Машина тертя СМЦ-2, будучи модифікацією машини тертя Амслера, призначена для випробування матеріалів на знос і визначення їх фрикційних властивостей в умовах тертя ковзання і тертя кочення при нормальних температурах для модельних трибосистем [10].

Пару тертя диск-диск використовують для моделювання роботи трибосполучень з лінійним контактом елементів, таких як колесорельс (наприклад кранові ходові колеса, що переміщаються по рейці) або зубчасте зачеплення. При взаємному обкатуванні взаємодіючих дисків з деяким прослизанням в зоні їх

контакту виникають умови навантаження матеріалу, відповідні нагружению матеріалу зубчастого колеса в будь-якій точці лінії контакту (зачеплення).

Пара тертя диск-колодка використовується для моделювання роботи трибосполучень сухого і граничного тертя (гальмівні колодки і ін.).

При використанні модельної трибосистеми втулка-вал можливо дослідження гідродинамічної опори тертя. Схема моделей взаємодії пар тертя, реалізованих на машині тертя СМЦ-2 наведено на рисунку 1.1, де а – диск-диск; б – диск-колодка; в – вал-втулка; 1 – зразок; 2 – контртіло; P – навантаження; ω_1, ω_2, ω – кутові швидкості обертання.

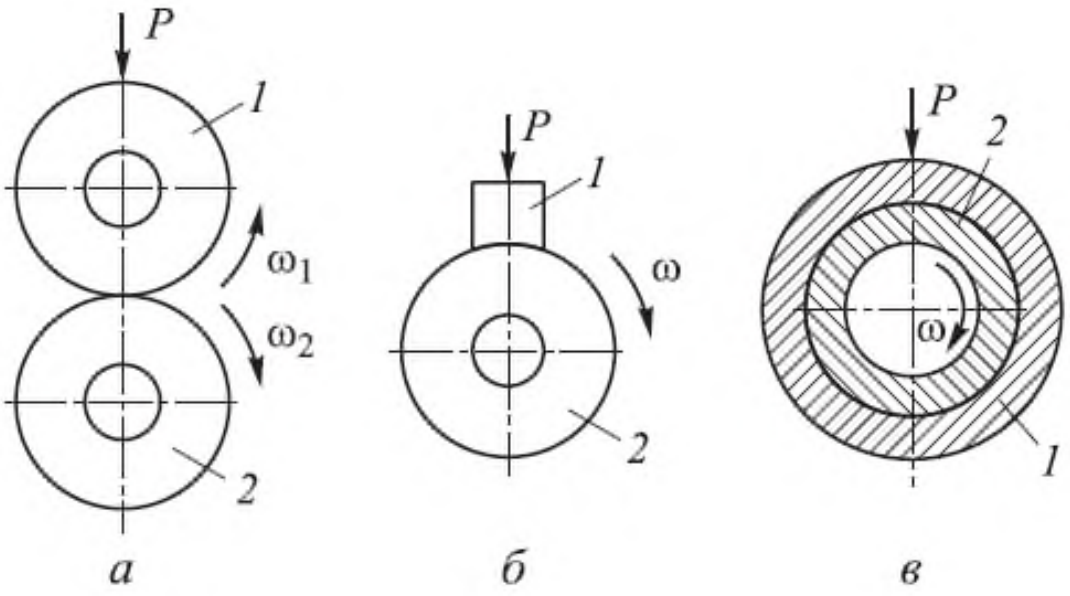

Рис. 1.1. Схема моделей взаємодії фрикційних пар, реалізованих на машині тертя СМЦ-2

Трибометр укомплектований пристроями для тарування його силових систем, а також для проведення випробувань елементів модельних трибосистем диск-диск і диск-колодка в рідких середовищах передбачено використання спеціальних камер.

На рисунку 1.2. приведена схема випробувань на зношування з початковим лінійним контактом, здійснювана на машині тертя СМЦ-2 з використанням циліндричних зразків. Основні елементи, що наведені на рисунку 1.2: 1, 6 – вали; 2 – гайка; 3 - шайба; 4, 5 – зразки; 7 – стопор; 8 – вана; 9, 10 – термопари; 11, 12 – потенціометри КСП-4.

Машину тертя СМЦ-2 використовують також для дослідження реологічних властивостей індустріальних мастил.

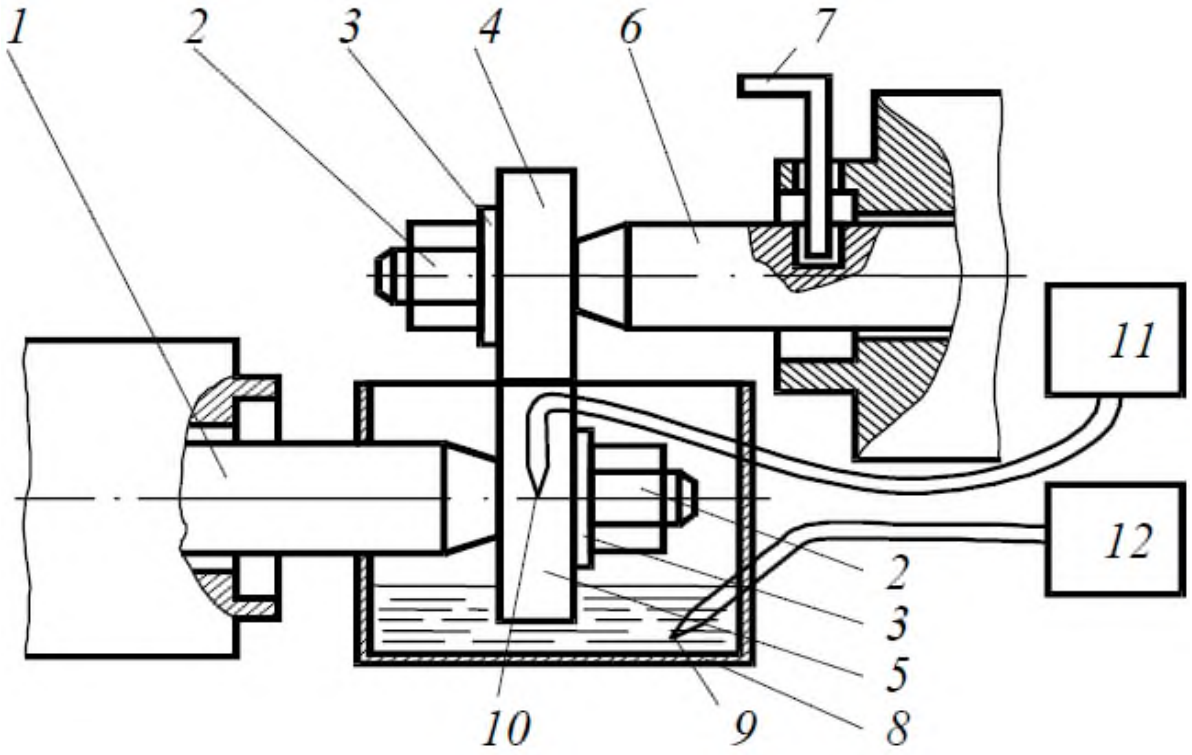

Рис. 1.2. Схема випробування мастильних матеріалів на машині тертя СМЦ-2

Індустріальні масла являють собою великий асортимент мастильних матеріалів, призначених для зменшення тертя, зношування та запобігання задертостям тертьових поверхонь різних машин та механізмів промислового обладнання: метало- та деревообробних верстатів, пресів, контрольно-вимірювальних приладів, насосів гідросистем тощо. Одночасно вони повинні відводити теплоту з трибосистеми, захищати деталі від корозії, очищувати тертьові поверхні від забруднення, служити ущільнюючим засобом, запобігати утворенню стійких емульсій з водою або мати здатність емульгувати, добре фільтруватися, бути нетоксичними, не мати неприємного запаху та ін.

В умовах застосування мастильні індустріальні оливи піддаються впливу високих температур і тисків, контактують з різними металами, повітрям, водою та агресивними середовищами. У результаті чого вони окислюються, що призводить до підвищення їхньої в’язкості, кислотного числа та корозійної агресивності, забруднення продуктами зношення. Внаслідок цього посилюється абразивне

зношування у трибосистемах і погіршується фільтрування масла. В маслах з'являються продукти деструкції, які викликають зниження в'язкості та температури спалаху, накопичується вода та ін. Індустріальні масла, які застосовуються у трибосистемах, можна розглядати як свого роду конструкційний матеріал, властивості якого у ряді випадків впливають на працездатність поверхні тертя не менше, ніж властивості матеріалу, з якого виготовлені деталі, що змащуються.

Експлуатаційні властивості індустріальних масла істотно впливають на втрати потужності на тертя, зношення тертьових деталей, плавність руху при малих швидкостях ковзання, стабільність окиснення та корозії, емульгованість, піноутворення і деаерацію. Найважливішими з них є: мастильні властивості, здатність стабілізувати коефіцієнт тертя, забезпечуючи при цьому плавне стійке ковзання і демпфірування, антиокислювальну стабільність, захисні, деемульгуючі, протипінні, деаераційні, екологічні властивості. Умова тертя є визначальною при виборі оливи. Мастильні вузли та деталі, залежно обладнання, яке застосовується, значно відрізняються умовами роботи, температурними, навантажувальними та іншими характеристиками.

1.4. Опис універсальної машини тертя типу СМЦ-2 та її автоматизованої системи керування та вимірювання

Для дослідження формованих результуючих сигналів, які виникають при випробуванні дослідних пар тертя (ПТ) використовувався апаратно-програмний комплекс, що представляє собою автоматизований стенд для проведення випробувань на тертя та зношування на базі універсальної машини тертя типу СМЦ-2 (рис. 1.3).

Комплект серійної апаратури та спеціально розроблене програмне забезпечення випробувального стенду дозволяє проводити керування режимами навантаження дослідної ПТ, а також проводити вимірювання таких трибологічних параметрів як температура в зоні тертя, коефіцієнт тертя (момент тертя) та будь-які інші. До основних елементів апаратно-програмного комплексу (АПК) відносять (рис. 1.3):

1 – каретка; 2 – механізм навантаження; 3 – вузел нижнього зразка; 4 – датчик; 5 – привід; 6 – пульт управління; 7 – триступеневий провідний шків; 8 – клинові ремені; 9 – ведений шків; 10 – редуктор; 11 – кінцевий вимикач; 12 – лічильник сумарного числа обертів нижнього зразка; 13 – показуючий і записуючий потенціометр.

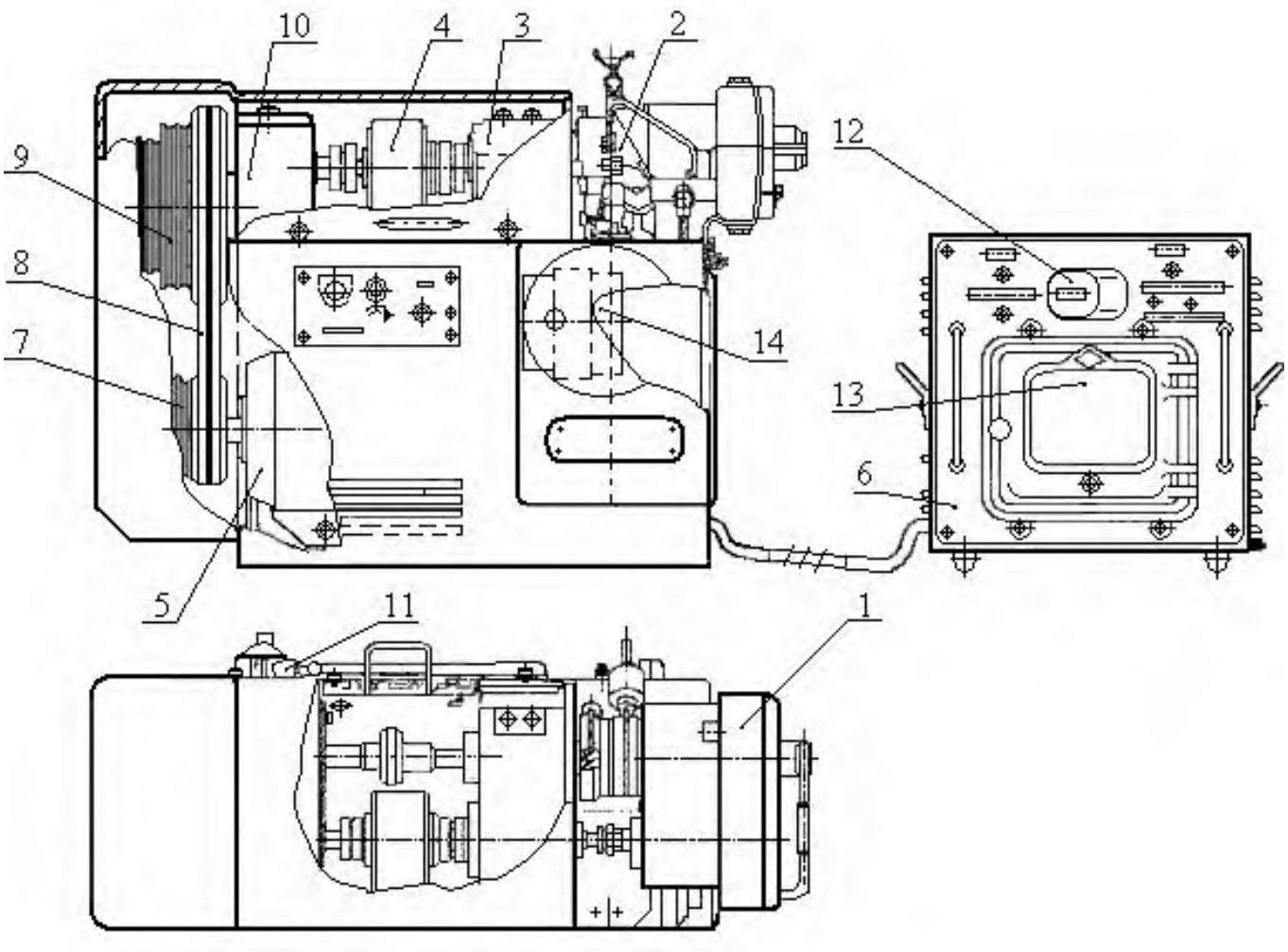

Рис. 1.3. Основні елементи апаратно-програмного комплексу для проведення випробувань на тертя та зношування дослідних зразків

Універсальна випробувальна машина СМЦ-2 призначена для дослідження процесів тертя та зношування металів, сплавів та жорстких конструкційних пластмас в приміщеннях лабораторного типу. Принцип дії випробувальної машини полягає в зношуванні дослідної пари зразків, які взаємодіють між собою з певною силовою дією. При випробуваннях один із дослідних зразків залишається нерухомим, а інший зразок обертається з заданою кутовою швидкістю.

Кінематична схема випробувальної установки СМЦ-2, на якій зображено основні елементи спеціалізованої машини, представлена на рисунку 1.4.

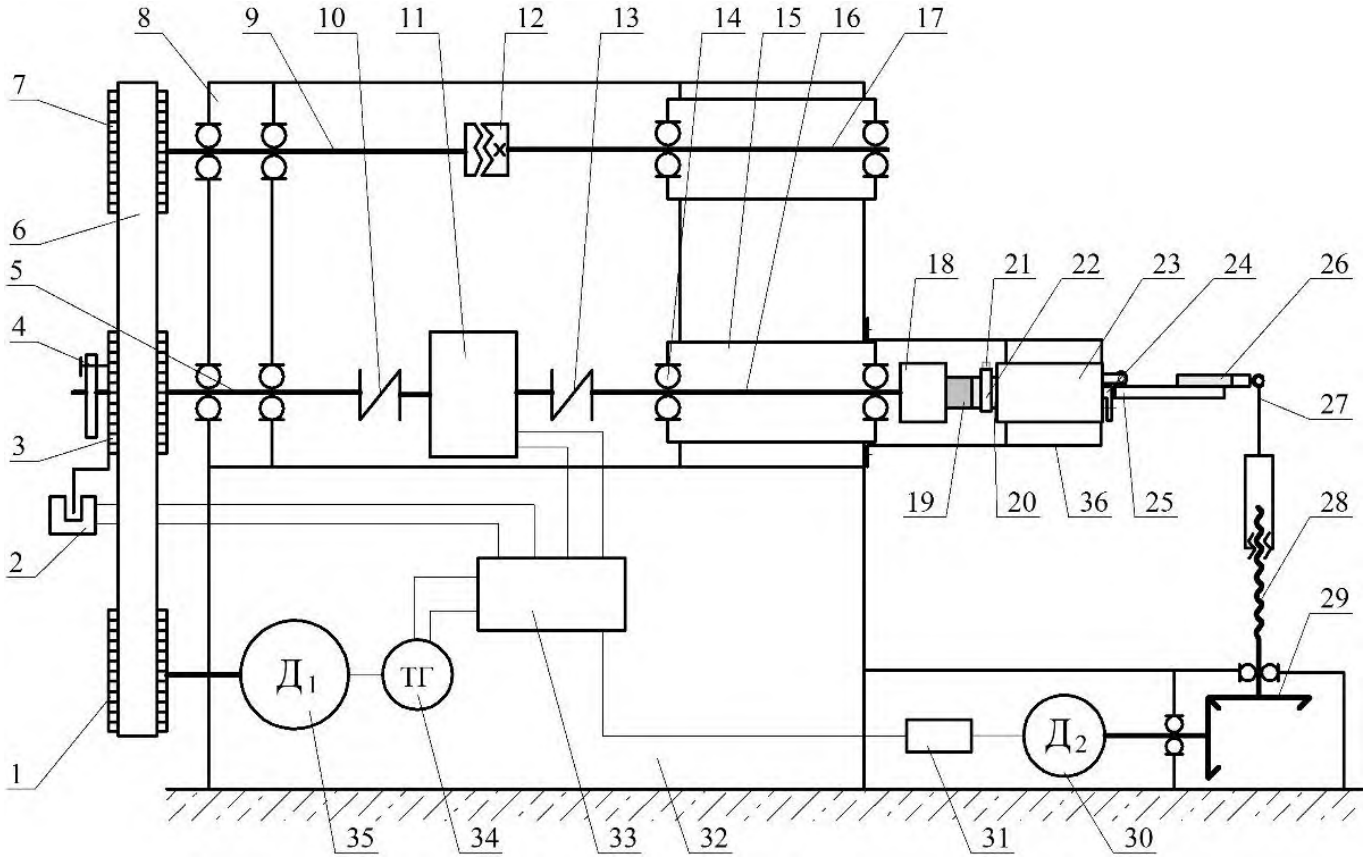

Рис. 1.4. Кінематична схема випробувальної машини СМЦ-2 для проведення випробувань на тертя та зношування дослідних зразків

Основні елементи машини встановлені на станині 32, всередині якої розміщені електродвигуни постійного струму 35 та 30, допоміжне електрообладнання та деталі, які дозволяють регулювати натяг ременя 6. Для надання дослідному зразку обертального руху використовується електродвигун 35, який обертає зубчастий ремінь 6, потім шківи 1, 3 та 7. Через запобіжний штифт 4, вал 5, муфту 10, датчик моменту 11, муфту 13 обертання передається на вал бабки 16, де встановлюється змінний патрон 18 для закріплення рухомого зразка 19.

Шків 7 через вал 9, кулачкову муфту 12 обертає вал 17, який використовується для обертання зразка за схемою випробувань «ролик-ролик». Нерухомий дослідний зразок 20 встановлюється в патрон, який закріплений на обоймі 23.

Силова взаємодія зразків забезпечується навантаженням, яке передається через важільний механізм, що приводиться в дію від електродвигуна 30. Принцип дії механізму осьового навантаження полягає в наступному: обертання від електродвигуна 30, який керується реле-регулятором 31 системи керування, передається на вісь-гвинт 28 через конічну передачу 29. Тяга 27 переміщується у вертикальному напрямку вниз та через тензодатчик 26 та важіль 25 діє на підкладку 24, яка з'єднана з обоймою 23. Величина прикладеного осьового навантаження визначається за допомогою тензодатчика 26, що трансформує деформацію під силою навантаження в аналоговий сигнал, що передається до комп'ютерної системи керування.

Випробувальна камера в якій встановлюються дослідні зразки складається з корпусу 15 та кришки 36. В передній отвір корпусу 15 встановлюється змінний шпиндель 16 разом з радіальними підшипниками 14. На лівій частина змінного шпинделя 16 встановлюється півмуфта 13 для з'єднання з датчиком моменту 11. На правій стороні змінного шпинделя 14 встановлюється змінний патрон 18. Підшипники захищені від пилу та бруду лабіринтовими ущільнювачами. Верхня та бокові частини корпусу випробувальної машини СМЦ-2 закриті захисними кожухами.

Частота обертання дослідних зразків, а також загальна кількість обертів визначаються відповідно кінцевим вимикачем 2 та тахогенератором 34, формується аналоговий сигнал який передається до системи управління.

Реєстрація формованих результуючих сигналів АЕ здійснюється датчиком АЕ 21, який встановлюється на нерухомий дослідний зразок 20 через хвилевід 22. Блок 33 забезпечує перетворення аналогових сигналів в цифрові коди з їх подальшим аналізом.

Для контролю температури в парі тертя на машину встановлена термопара, що під'єднана до променевого осцилографа. До нього ж приєднаний вихід датчика моменту тертя. Осцилограф дозволяє вести безперервне спостереження за парою тертя. Логічна схема вимірювань температури наведена на рисунку 1.5, на якій зображені такі елементи: 1 – зразок «колодка»; 2 – зразок «диск»; 3 – термопара; 4 –

датчик моменту; 5 – з'єднувальні дроти; 6 – датчик вимірювання числа циклів роботи; 7 – осцилограф; 8 – потенціометр; 9 – лічильник числа циклів.

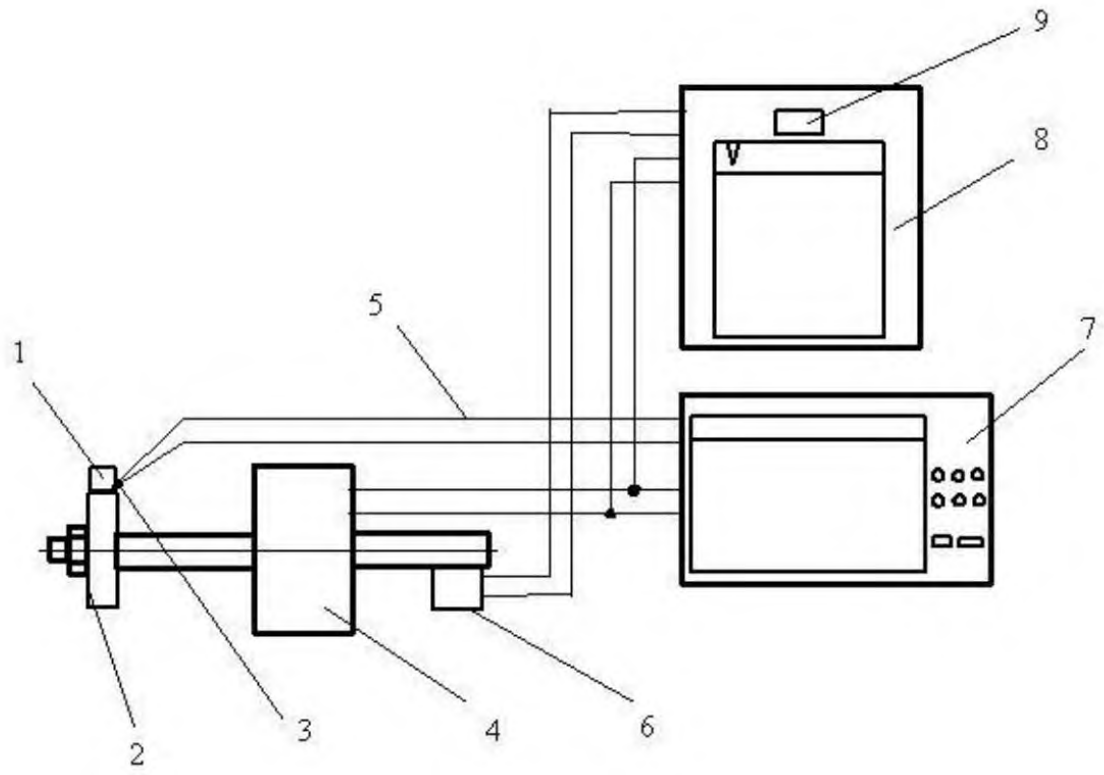

Рис. 1.5. – Схема безперервного вимірювання температури пари тертя

Структурна схема системи керування та вимірювання АПК представлена на рис. 1.6. Необхідна частота обертання дослідного зразка забезпечується регулятором напруги 6, який з’єднаний з електродвигуном постійного струму 7 типу П32УХЛ4.

Керуючий сигнал формується програмно в ПК 1 і через перетворювач 2 та з’єднувальну плату 4 поступає на регулятор напруги 6. Поточне значення частоти обертання електродвигуна 7 реєструється датчиком швидкості 5. Необхідне силове навантаження на дослідні зразки забезпечується блоком навантаження 19, який приводиться в дію від електродвигуна 20. Формований керуючий сигнал з ПК поступає до блоку навантаження 19 через перетворювачі 2, 17 та блок реле 18. Поточне значення прикладеного осьового навантаження реєструється тензодатчиком 15, інформація з якого поступає до ПК 1 через АЦП 14 та перетворювач 2 [11].

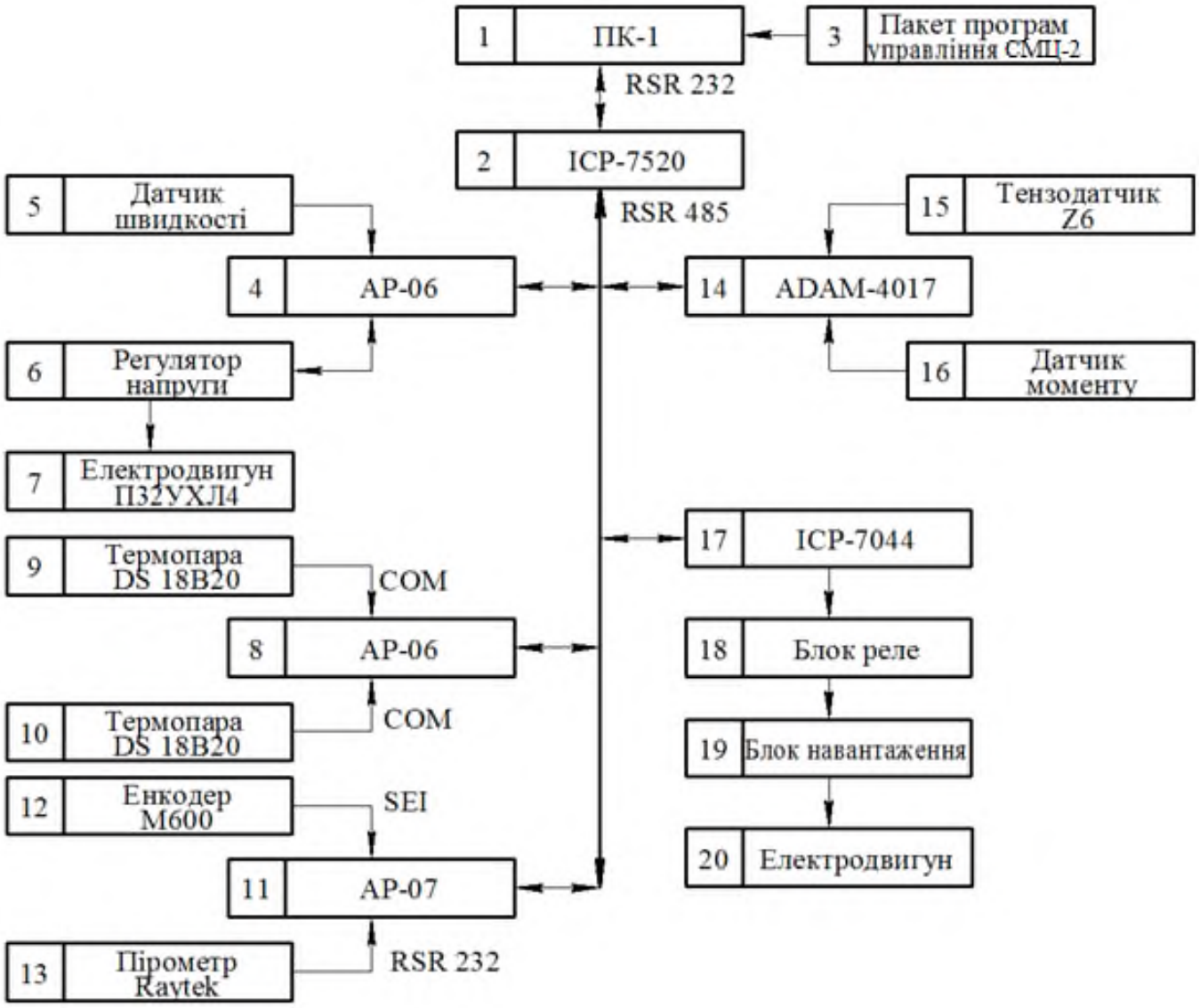

Рис. 1.6. Структурна схема автоматизованої системи керування та вимірювання СМЦ-2

Температурні показники навколишнього та змащувального середовищ реєструються, відповідно, термопарами 9 та 10. Інформація з термопар 9 та 10 поступає до ПК 1 через з'єднувальну плату 8 та перетворювач 2. Температура в зоні тертя реєструється пірометром 13, який знаходиться в близькій відстані від поверхонь фрикційного контакту. Інформація від пірометра 13 передається через з'єднувальну плату 11 та перетворювач 2 на ПК 1. Лінійні осьові переміщення, які виникають при випробуванні зразків реєструються датчиком переміщення 12. Інформація з датчика переміщення 12 передається через з'єднувальну плату 11 та перетворювач 2 на ПК 1.

Момент тертя, який виникає при обертанні рухомого зразка дослідної ПТ реєструється датчиком моменту 16, інформація з якого поступає через АЦП 14 та перетворювач 2 на ПК 1.

Програмне забезпечення комп'ютерної системи керування дозволяє:

– виводити інформацію про поточні значення вимірюваних параметрів датчиків та їх працездатність;

– будувати графічні часові залежності вимірюваних параметрів датчиків в режимі реального часу;

– встановлювати необхідні режими навантаження (швидкість обертання зразків та прикладене осьове навантаження) та створювати необхідні температурні умови випробувань;

– планувати досліди з зазначенням режимів навантаження зразків та часу початку випробувань.

1.5. Постановка завдання проектування

Метою даного дослідження є розробка апаратно-програмної системи керування машиною тертя СМЦ-2 для автоматизації триботехнічних випробувань на дослідження реологічних властивостей індустріальних мастил. Апаратно-програмна система складатиметься з двох логічних частин: керування та збору інформації. Розроблена система надасть можливість автоматизованого керування, вимірювання, обробки та збереження показників датчиків при проведенні експериментальних триботехнічних випробувань на дослідження реологічних властивостей на машині тертя СМЦ-2, що збільшить точність та якість проведених досліджень.

У відповідності до сформульованої мети у проекті поставлено наступні завдання:

– проаналізувати технології проектування автоматизованих систем керування та збору інформації;

– розробити структуру апаратно-програмної системи;

– розробити систему керування та збору інформації;

– протестувати роботу апаратно-програмної системи.

1.6. Висновки до розділу

1. Розглянуто основні принципи функціонування та класифікацію автоматизованих систем керування, переваги автоматизації систем керування. Досліджено класифікацію приладів контролю та автоматичного регулювання.

2. Описано реологічні властивості індустріальних мастил.

3. Досліджено процес проведення випробувань на тертя та зношування.

4. Описано спеціалізовану машину тертя СМЦ-2, її автоматизовану систему керування, встановлено її недоліки.

5. Поставлено завдання проектування.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ТА МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ МАШИНОЮ ТЕРТЯ СМЦ-2

2.1. Дослідження технологій проектування автоматизованих систем керування та збору інформації

Тенденції сучасного етапу розвитку систем керування та обробки сигналів свідчать про збільшення сфер їх застосування, таких, що мають опрацьовувати інтенсивні потоки інформації у режимі реального часу і задовольняти вимогам вартості, часу розробки, енергоспоживання і габаритів.

Для імплементації цих алгоритмів у комп’ютерних системах існує три базові способи: програмний, мікропрограмний або апаратний. Кожний з цих методів має свої унікальні особливості, переваги, недоліки.

До АСУ в цілому ставляться наступні основні вимоги:

– здійснювати управління над об’єктом управління в цілому в темпі протікання технологічного процесу і в вироблені і реалізації рішень по керуванню повинні брати участь засоби обчислювальної техніки і оператор;

– забезпечувати управління об’єктом відповідно до прийнятих критеріїв ефективності функціонування АСУ;

– виконувати всі покладені на неї функції з заданими характеристиками і показниками якості управління;

– володіти необхідним рівнем надійності;

– забезпечувати можливість взаємопов'язаного функціонування з системами управління суміжних рівнів ієрархії і іншими АСУ;

– відповідати ергономічним вимогам, що пред'являються до систем, зокрема до способів і формі. Подання інформації оператору, до розміщення технічних засобів і т.д.;

– володіти необхідними метрологічними характеристиками вимірювальних каналів;

– допускати можливість модернізації і розвитку в межах, передбачених технічним завданням на створення АСУ.

Використовуючи програмний спосіб реалізації алгоритмів керування та обробки сигналів, головним недоліком є той факт, що обчислювальні процеси найчастіше розгортаються у часі з великим об’ємом транспортування інформації між операційними пристроями та оперативним запам’ятовуючим пристроєм. Перевагами програмного способу є те, що цей метод має найменші час розробки та трудоємність, процес модифікації алгоритму є найшвидшим і найлегшим серед інших способів, перелічених вище [12].

Мікропрограмний метод імплементації алгоритмів визначає їхнє розгортання в часі та просторі. Використовуючи мікропрограмування можна отримати доступ до мікропрограм процесора за допомогою використання таких засобів, як логічні матриці, постійна пам’ять та оперативні запам’ятовуючі пристрої, що використовується в ролі основної пам’яті мікропрограм. Головним недоліком цього методу є набагато більший час розробки, необхідність наявності експертизи в домені проблеми, що має бути вирішена та повноцінного розуміння внутрішньої структури процесора та системи його команд. Порівнюючи мікропрограмні та програмні засоби імплементації алгоритмів перші є набагато швидшими.

Завдяки рівню сучасного розвитку інтегральних технологій існує можливість делегувати реалізацію алгоритмів на апаратні засоби. Такий метод реалізації передбачає використання спеціалізованого обладнання завдяки чому системи управління стають більш модульними та децентралізованими, проте це збільшує кількість проміжного пересилання інформації в процесі роботи алгоритму.

Для отримання максимальної швидкодії від апаратних засобів, що реалізують алгоритми управління та обробки сигналів рекомендується використовувати алгоритмічні структури. Алгоритмічними вважаються структури, що синтезовані у результаті використання принципу адекватного апаратного відображення потокових графів алгоритмів управління та обробки сигналів на комбінаційну матрицю,

процесорні елементи якої реалізують функціональні оператори та з’єднані між собою відповідно з потоковим графом алгоритми, в основі структурної організації апаратних засобів [13].

На практиці, при реалізації автоматизованих систем управління реального часу окремі варіанти реалізації алгоритмів майже не використовуються, натомість застосовуються комбіновані підходи, отримуючи лише переваги окремих методів. Вибір основного методу імплементації алгоритмів у АСУ реального часу виконується враховуючи вимоги забезпечення реалізацією алгоритму режиму реального часу та високої ефективності використання обладнання. У автоматизованих системах використовуються розпаралелювання та конвеєризація процесів обробки з метою забезпечення управління та обробки сигналів у режимі реального часу.

Існує три основні варіанти побудови автоматизованої системи управління та обробки сигналів у реальному часі:

– на основі універсальних і функціонально-орієнтованих мікропроцесорів, шляхом розробки спеціалізованого програмного забезпечення;

– на основі універсального обчислювального ядра доповненого базовими апаратно-програмними компонентами, які реалізують часомісткі алгоритми функціонування штучної нейронної мережі;

– у вигляді спеціалізованої алгоритмічної системи, архітектура та організація обчислювального процесу в якій відображає структуру алгоритму розв’язання задачі [14].

Варіант побудови АСУ на база мікропроцесорів є найбільш доступним варіантом. Головна його перевага – швидкість розробки програмного забезпечення (ПЗ) за рахунок використання існуючих бібліотек. Недоліком можна вважати порівняно невисоку швидкодію в системах надзвичайно великої інтенсивності вхідних повідомлень.

Варіант доповнення обчислювального ядра апаратно-програмними компонентами передбачає комбінування спеціальних та універсальних засобів, що робить його перспективним напрямом тому, що поєднання такого типу здатне забезпечити високу результативність використання обладнання у системах реального

часу при опрацюванні великих потоків інформації спеціалізованими алгоритмами. Сам процес розробки апаратних засобів з визначеними технічними характеристиками складається з доповнення обчислювального ядра додатковими компонентами.

Третій варіант орієнтується на роботу з високоінтенсивними потоками даних за рахунок використання складних алгоритмів. Висока продуктивність досягається ефективністю використання обладнання за рахунок узгодження інтенсивності вхідних потоків даних та обчислювальної здатності апаратної частини. В основному, для побудови обчислювальних потужностей використовують програмовані логічні інтегральні схеми тому, що це дозволяє динамічно модифікувати реалізацію алгоритму за невеликий проміжок часу.

Розробку та модернізацію функціоналу АСУ рекомендується виконувати покладаючись на інтегрований підхід, який включає актуальні методи управління і обробки сигналів та їх алгоритми, алгоритмічні, програмні, апаратні засоби та сучасну елементну базу, методи та засоби автоматизованого проектування апаратного і програмного забезпечення АСУ реального часу.

Принципи, що дозволяють зменшити вартість, терміни виконання і розширити галузі застосування апаратних засобів АСУ та обробки сигналів мають бути покладені в основу їх побудови. Забезпечення даних вимог може бути виконане при використанні наступних принципів побудови:

– модульності, що передбачає розробку компонентів АСУ у вигляді функціонально завершених пристроїв (модулів), що мають вихід на стандартний інтерфейс;

– узгодженості інтенсивності надходження даних з обчислювальною здатністю апаратних засобів АСУ;

– конвеєризації та паралелізму роботи з даними;

– відкритості програмного забезпечення, що передбачає можливості нарощування та його вдосконалення, максимального використанням стандартних драйверів та програмних засобів;

– спеціалізації та адаптації апаратно-програмних засобів до структури алгоритмів обробки та інтенсивності надходження даних;

– програмованості архітектури шляхом використання репрограмованих логічних інтегральних мікросхем;

– використання паралельної пам’яті для зберігання інформації та обміну між компонентами системи [15].

При визначенні та розробці методів і алгоритмів обробки даних враховуються вимоги і характеристики, але визначальним є забезпечення обмежень. Інформаційні, операційні, точність – характеристики, що використовуються для оцінки розроблених алгоритмів. До інформаційних характеристик відносять: кількість констант, вхідних, проміжних та вихідних даних, кількість каналів та їх розрядність, види операцій. Операцайні характеристики дозволяють оцінювати час, необхідний для реалізації та обчислювальну здатність. До характеристик точності відносять розрядність операційних пристроїв, способи округлення. При розробці АСУ обов’язковою умовою є забезпечення узгодженості інтенсивності надходження інформації з інтенсивністю їх обчислення, на всіх етапах обробки цієї інформації в комп’ютерній системі.

В АСУ реального часу висока ефективність використання обладнання досягається мінімізацією витрат обладнання на реалізацію обробки сигналів при забезпечені реального часу. Перехід від алгоритму розв’язання задачі в режимі реального часу до структури АСУ формально зводиться до мінімізації витрат обладнання.

Основними шляхами мінімізації затрат на обладнання при проектуванні спеціалізових комп’ютерних систем управління та обробки сигналів у реальному часі є:

– вибір ефективних методів і алгоритмів розв’язання задач і реалізації функціональних операторів;

– зменшення розрядності операційних пристроїв, ємності пам’яті, кількості і розрядності каналів передачі даних;

– узгодження інтенсивності надходження даних із обчислювальною здатністю апаратних засобів АСУ [16].

Однією з найбільш відповідальних задач при проектуванні АСУ є перехід від алгоритму розв’язання задачі до власне структури АСУ. Алгоритми розв’язання задач можна описувати через залежність між входом і виходом або детально пояснюючи їх внутрішню структуру. Подання алгоритмів у формі стандартного математичного запису не дозволяє достатньо повно оцінити можливості паралелізації та знайти способи їх ефективної реалізації на існуючих обчислювальних засобах або створити нові ефективні структури. Тому для вибору ефективного варіанту реалізації алгоритму може застосовуватись подача його у формі, яка одночасно відображає просторові та часові характеристики.

2.2. Модель модернізованої автоматизованої системи управління СМЦ-2

Враховуючи інформацію, отриману під час досліджень, а саме опис принципів побудови автоматизованих систем управління та збору інформації, та постановку задачі в конкретних умовах доцільним є модернізація системи шляхом заміни великої кількості модулів на сучасніші, дешевші, менші за габаритами аналоги. Основними критеріями вибору платформи для модернізації є: популярність серед користувачів, що дозволяє швидко знаходити відповідь на питання та прискорює час розробки за рахунок наявної програмної бази, вартість, відкритість, кількість фізичного простору для реалізації, час автономної роботи (низьке енергоспоживання).

Плати для розробки можна розбити на дві великі категорії: плати на мікроконтролері та одноплатні комп’ютери. Порівняльна характеристика цих плат наведена в таблиці 2.1.

Плата – це друкована плата з певною схемою та обладнанням, що полегшує безліч експериментів на основі додатків, які можливі з особливостями цієї плати мікроконтролера; поєднання мікропроцесора, пам'яті, інтегральних схем (IC) та периферійних пристроїв, наприклад, *USB*-порт, АЦП, послідовний порт, *Ethernet, RTC*, регулятори напруги, слот мікросхеми пам'яті тощо з функціями скидання.

Такі типи плат не тільки рятують інженерів або розробників від витрати часу на з’єднання зовнішніх перемичок, що також економить час. Плати мікроконтролера слід вибирати на основі використовуваного процесора, типу шини даних, простору пам'яті, операційної системи та портів вводу-виводу.

Таблиця 2.1

Порівняльна характеристика плат

Параметр	Мікроконтролер	Одноплатний комп’ютер
Продуктивність	1 ядро, десятки-сотні *МГц*, десятки *КБ* оперативної пам'яті, десятки-сотні *КБ* постійної пам'яті.	1 або більше ядер, сотні-тисячі *МГц*, сотні *МБ* оперативної пам'яті, гігабайти постійної пам'яті.
Багатозадачність	Ні. Але можна емулювати.	Так. Управляється ОС.
Зручність роботи з інтернетом	Зазвичай потрібні додаткові модулі і глибоке знання протоколів.	Легко підключається з коробки, мережевий модуль зазвичай вже на борту.
Тривалість роботи від батареї	Споживає одиниці-десятки *мА*. Можливі тижні роботи від батарейок.	Споживає сотні-тисячі *мА*. Заряду великого акумулятора вистачить від сили на десяток годин.
Швидкість реакції в проектах критичних до часу	100 % контроль над часом і тривалістю подачі сигналів.	Через багатозадачності критичний процес може «проспати» свій час.
Вибір мов програмування	Обмежений. Найчастіше *C/C++*.	*Python*, *JavaScript*, *Bash* і десятки інших: будь-які доступні в ОС.
Можливості для роботи з відео, комп'ютерним зором	Не вистачить потужності.	*OpenCV*, апаратні видеокодеки, *HDMI*-вихід.
Можливості для роботи зі звуком	На потужних мікроконтролерах можливий синтез звуку. Для роботи з *MP*3/*OGG*/*WAV* потрібні додаткові модулі.	Підтримка *MP*3/*OGG*/*WAV* на рівні ОС. Аудіовихід *HDMI* і/або роз'єм 3,5 *мм*.

Як видно з таблиці 2.1 одноплатні комп’ютери краще майже у всіх показниках, крім часу автономної роботи від батареї, також для виконання задачі управління та збору інформації СМЦ-2 операційна система *Linux* на проміжному пристрої є, скоріш надлишковістю ніж перевагою, тим більше, що вона контролює всі процеси і через це, навіть при додатковому налаштуванні ОС, пріоритетний процес може чекати часу свого виконання. Всі ті переваги, які надає одноплатний комп’ютер є надлишковими, а час автономної роботи є критичним.

Для вибору оптимальної плати необхідно навести порівняльну характеристику найновіших плат, які вже є на ринку. Характеристики наведені в порівняльній таблиці 2.2.

Таблиця 2.2

Порівняльна таблиця мікроконтролерів

Параметр	*Arduino Uno*	*mbed NXP LPC1768*	*Intel Galileo*
Робоча напруга (В)	5	4.5-9	3.3
Мікроконтролер	*AVR ATmega328*	*ARM Cortex-M3*	*Intel Quark SoC X1000*
Довжина машинного слова (біт)	8	32	32
Середовище програмування	*IDE*	Онлайн *IDE*	*IDE*
Тактова частота (МГц)	16	100	400
Флеш пам’ять (КБ)	-	512	512
SRAM (КБ)	2	64	512
EEPROM (КБ)	1	512	11
Кількість цифрових контактів входу/виходу	14	20	14
Кількість аналогових контактів входу/виходу	8	8	8
Залежність від платформи	Відсутня	Відсутня	Відсутня
Кількість таймерів	2	4	4
Ethernet (МБ/сек)	Відсутній (досягається використанням зовнішнього модулю)	10/100	10/100

Продовження таблиці 2.2

Параметр	*Arduino Uno*	*mbed NXP LPC1768*	*Intel Galileo*
Wi-Fi	Відсутній (досягається використанням зовнішнього модулю)	Відсутній (досягається використанням зовнішнього модулю)	Відсутній (досягається використанням зовнішнього модулю)
SPI	Присутній	Присутній	Присутній
I2C	Присутній	Присутній	Присутній
USB	Присутній	Присутній	Присутній (клієнт і хост)
Приблизна вартість (гривень)	800	1500	3000

Найпопулярнішою і найдешевшою є апаратно-обчислювальна платформа (АОП) *Arduino*. *Arduino* являє собою досить простий інструмент для створення електронних пристроїв і втілення в життя різних ідей. Це платформа побудована на друкованій платі з інтегрованим середовищем для написання програмного забезпечення. В основі апаратної частини лежить мікроконтролер сімейства *ATMega* і мінімально необхідне для роботи вбудоване обладнання [17].

Arduino може приймати цифрові і аналогові сигнали з різних пристроїв і має можливість управління різними виконавчими модулями.

Цей мікроконтролер дуже популярний серед любителів збірки саморобних пристроїв різних вікових груп.

Популярності сприяє відносно проста робота з апаратно-обчислювальною платформою, великий набір різних компонентів для збірки пристроїв і мережа Інтернет, де користувачі діляться своїми напрацюваннями і спільно вирішують різні проблеми в при вирішенні будь-яких рішень. Є багато інших переваг.

По-перше, це низька вартість. Плати *Arduino* відносно дешеві в порівнянні з іншими платформами.

По-друге, це кросплатформеність. З *Arduino* можна працювати на системах під управлінням ОС *Windows, Mac OS* і *Linux*.

По-третє, це проста і зрозуміла середовище програмування. Середовище розробки спроектована для новачків, не знайомих з розробкою програмного забезпечення. Однак це не заважає досвідченим користувачам створювати і досить складні проекти. Середовище являє собою додаток, яке включає в себе редактор коду, компілятор і спеціальний модуль для прошивки плати. Мова програмування, що використовується в *Arduino*, є реалізацією *Wiring*. Строго кажучи, це *C*/*C++*, доповнений деякими бібліотеками.

По-четверте, це можливість апаратного розширення. Можливості плат *Arduino* можна розширити за допомогою особливих мікросхем, які іменуються «шилд» (від англ. *Shield*). Шилд встановлюються поверх основної плати та дають нові можливості. Так, наприклад, існують плати розширення для підключення до локальної мережі та інтернету (*Ethernet Shield*), для управління потужними моторами (*Motor Shield*), для отримання координат і часу з супутників *Global Positioning System* (модуль *GPS*) і багато інших.

Arduino – апаратно-обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролеру з елементами вводу/виводу та середовище розробки вбудованих програм, що називаються скетчами, для мікроконтролера на мові програмування, що є спрощеною підмножиною *C++*.

За допомогою *Arduino* можна розробляти різні інтерактивні пристрої, вимірювальні прилади, обробляти дані датчиків і перемикачів, управляти двигунами і т.д.. Також *Arduino* використовується для створення електронних пристроїв з можливістю прийому сигналу від різноманітних цифрових і аналогових датчиків, що можуть бути до нього підключені, та керування різноманітними керуючими приладами. Проекти приладів, в основі яких розміщено *Arduino*, можуть працювати самостійно або взаємодіяти з програмним забезпечення на комп’ютері.

Порівнюючи *Arduino Uno* з *LPC1768* і *Galileo Arduino* програє у всьому крім вартості та часу автономної роботи. Головним недоліком *Arduino*, роблячим його не дуже підходящим для створення чутливої до часу АСУ СМЦ-2 є те, що *Arduino* має 8-ми розрядні таймери, що означає малу роздільну здатність і затримки при

переповненні лічильника на обробку переривання. Натомість *LPC1768* та *Galileo* мають чотири 32-розрядні таймери, що дозволяє їм керувати об'єктом керування з високою точністю, оскільки керуючі сигнали будуть посилатись з високою точністю та без затримок. Велика кількість таймерів дозволяє використовувати незадіяні таймери для широтно-імпульсної модуляції, що застосовується для управління електродвигунами СМЦ-2.

Обираючи між *mbed LPC1768* та *Intel Galileo* вибір стає очевидним. Найбільшою перевагою *LPC1768* над *Galileo* є загальна кількість контактів, що дає змогу збільшувати кількість обладнання прив'язаного до АСУ без сильного ускладнення апаратної підсистеми. В бік *LPC1768* схиляють також наступні параметри:

– процесор *ARM Cortex-M3*, який набагато краще підійде для вирішення проблеми управління машиною тертя оскільки забезпечує високу швидкість обробки даних за рахунок високої обчислювальної здатності, має низький рівень споживання електроенергії та забезпечує високоточний контроль за часом;

– платформа *mbed*, що дозволяє розробляти рішення в рази швидше ніж на платформах конкурентів (за рахунок того, що пристрої *mbed* створені для прототипування інтернету речей і оснащені дуже потужною високорівневою бібліотекою, наприклад, щоб відправити *Ethernet*-кадр необхідно написати 5 рядків коду);

– менша ціна.

До мікроконтролера можна підключити велику кількість пристроїв, що полегшує розробку апаратної підсистеми. Підключення через USB служить двом цілям – програмування контролера та його живлення.

Враховуючи використання платформи *mbed LPC1768*, структурна схема, що зображена на рисунку 1.3, може бути спрощена, за рахунок використання сучасної елементної бази. Старі аналогові датчики замінюються на нові, цифрові, не потребуючи аналого-цифрових перетворювачів. Модернізована структура зображена на рис. 2.1.

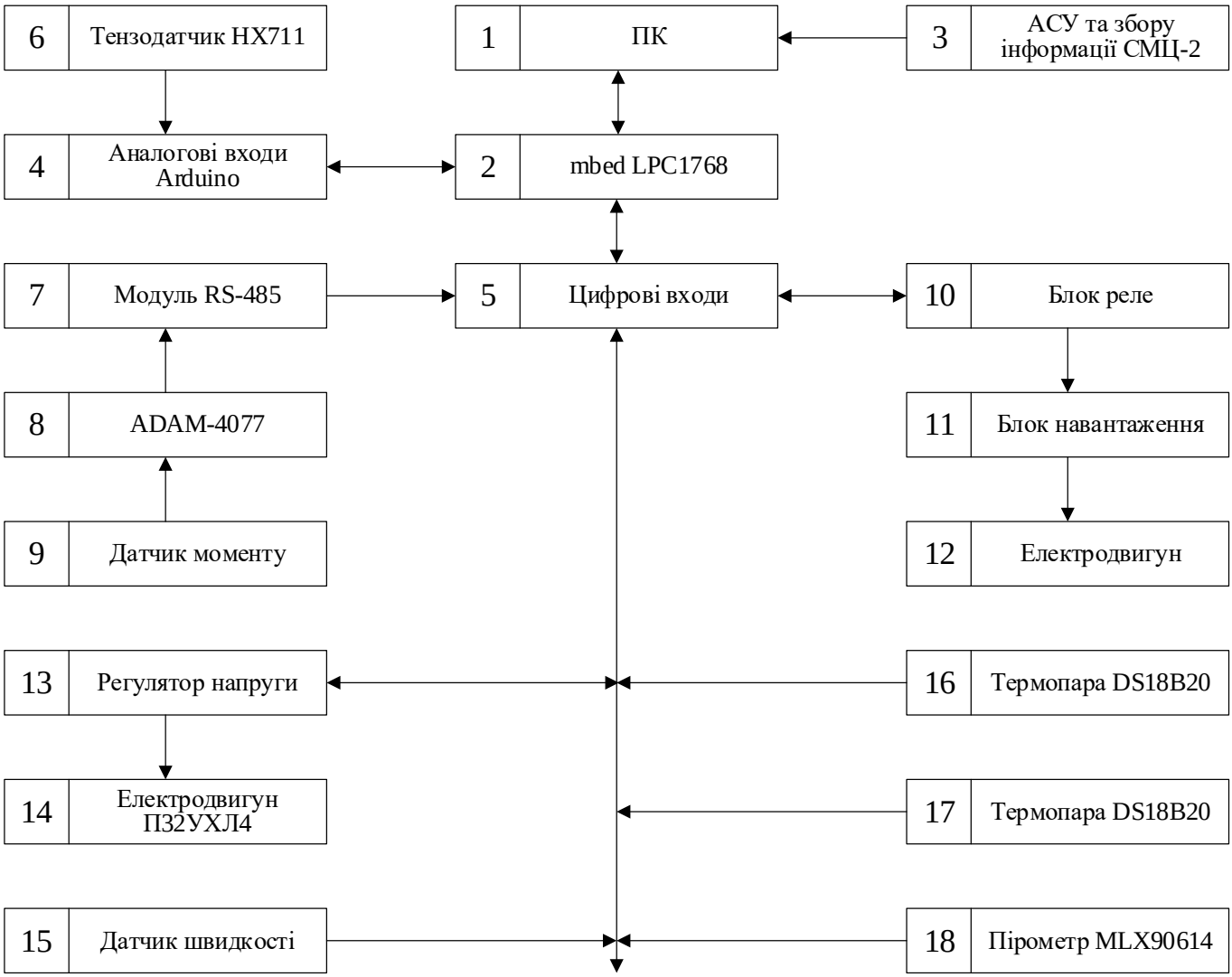

Рис. 2.1. Модернізована структура АСУ СМЦ-2 (схема структурна)

2.3. Логічна структура

Після визначення апаратної складової АПК потрібно визначити вимоги до програмної складової. Програмна складова АСУ і збору інформації СМЦ-2 поділяється на дві окремі частини. Перша частина – програма мікроконтролеру *LPC1768*, друга – основна логіка та графічний інтерфейс оператора АСУ.

Завданнями мікроконтролера є прийняття керуючих команд оператора, оброблення і керуючих сигналів на потрібний вузол, збір та обробка інформації з сенсорів та передача цієї інформації на ПК. Оскільки деякі сенсори передають інформацію у вигляді, не зручному для відображення на графічному інтерфейсі, АОП *LPC1768* повинна забезпечувати первинну обробку інформації.

Для забезпечення принципу модульності апаратно-обчислювальна платформа *LPC1768* має залишати інтерфейс незмінним, незважаючи на зміну апаратного обладнання та формату його повідомлень. Саме тому будь-яка обробка даних з сенсорів, для приведення їх у формат, що є зрозумілим для програмного забезпечення на ПК, та команд оператора, для приведення у вигляд, зрозумілий конкретному пристрою, якому адресоване це повідомлення, має відбуватись мікроконтролером.

Завданнями автоматизованого робочого місця є відображення інтерфейсу користувачу та обробка взаємодії користувача з інтерфейсом, відправка команд на мікроконтролер та отримання показників датчиків з мікроконтролера. Важливою частиною АСУ є збір, обробка та збереження інформації сенсорів, можливе графічне відображення цієї інформації в режимі реального часу. Найважливішою задачею, що має вирішуватись автоматизованим робочим місцем (АРМ) є керування машиною тертя, а саме запуск двигуна, налаштування його параметрів, таких як швидкість, навантаження та інші.

Апаратна платформа виконує функції централізованого збору інформації з підключених сенсорів та передачу цих даних на ПК, отримання і обробку команд оператора та передачу цих команд на відповідний пристрій. Також мікроконтролер може виконувати обробку даних, наприклад, перетворення аналогового сигналу на цифровий, за допомогою АЦП, переведення цих даних в потрібний формат.

Для вивчення вимог до автоматизованого робочого місця доцільним є використання діаграм прецедентів *UML*, вони дозволяють наочно описати роботу користувача з додатком. Розроблена діаграма прецедентів зображена на рисунку 2.2.

Як видно з діаграми, загальними вікнами роботи з програмою є вікно перегляду поточних показань датчиків, меню керування СМЦ-2 та меню роботи з планом виконання робіт. Для початку обміну даними з АОП має відбутись сканування фізичних портів або ж транспортних портів на предмет наявності повідомлень ініціалізації від мікроконтролера. Для забезпечення принципу декомпозиції вибір технології передачі даних не має впливати на основну частину ПЗ. Отже має бути розроблений додатковий модуль підключення *LPC1768*.

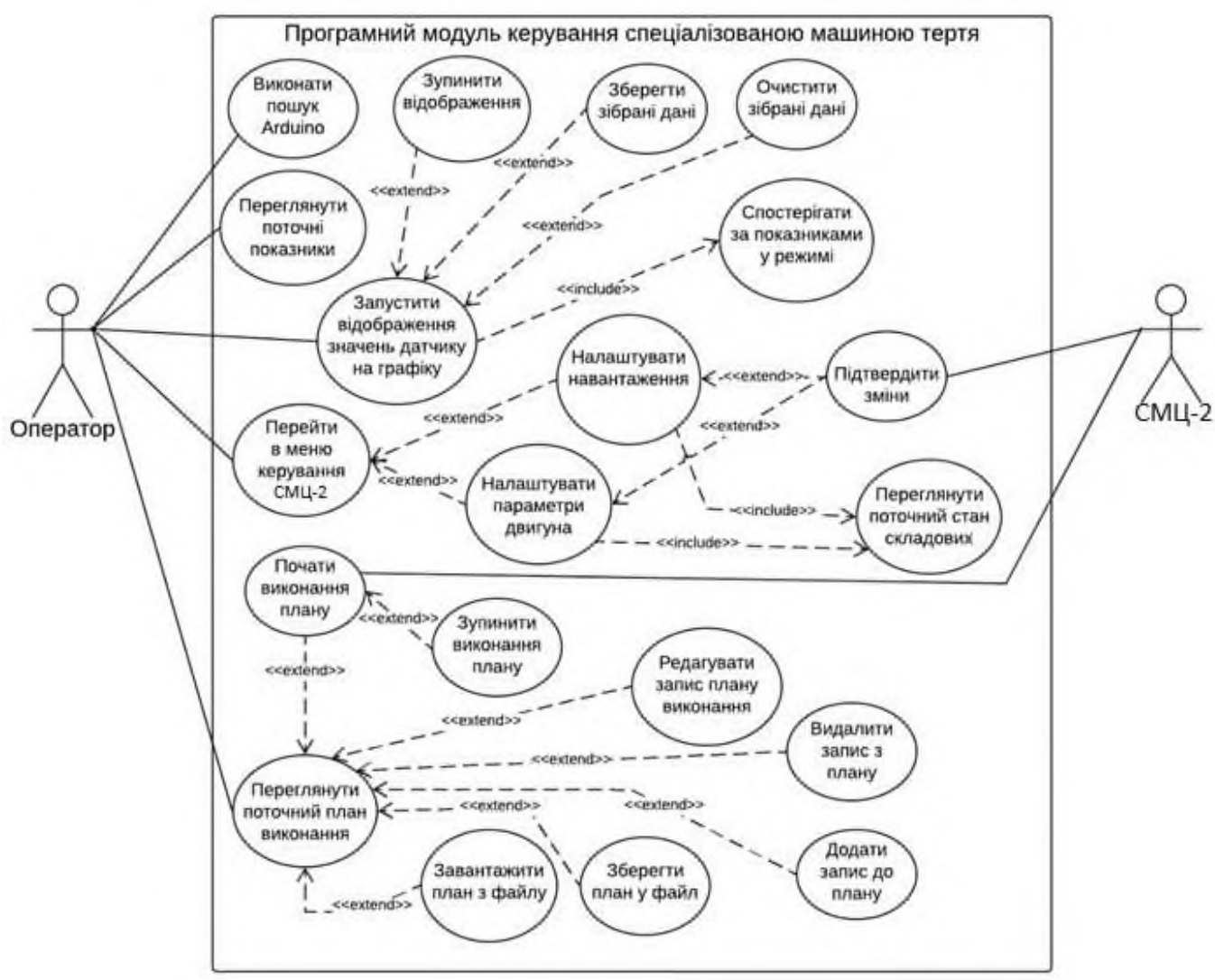

Рис. 2.2. Діаграма прецедентів

2.4. Висновки до розділу

1. Проаналізовано технології та методи проектування автоматизованих комп'ютерних систем управління. Описано принципи побудови, що дозволяють знизити вартість та скоротити час розробки. Для розробки автоматизованої системи керування обрано комбінований підхід з використанням переваг програмного та апаратного способу.

2. Проведено порівняння представників сучасної елементної бази мікроконтролерів на платі та одноплатних комп'ютерів. Перевагу надано мікроконтролерам, а саме *mbed LPC1768* на базі кристалу *ARM Cortex-M3*, для реалізації централізованого керування периферією.

3. Розроблено модернізовану структуру автоматизованої системи керування СМЦ-2 враховуючи використання *mbed LPC1768*, як інтерфейсу між периферією та ПК. Створено структурну схему модернізованої структури АСУ.

4. Розроблено логічну схему у вигляді діаграми прецедентів.

РОЗДІЛ 3
ОПИС ПРОЦЕСУ РОЗРОБКИ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВАЛІННЯ МАШИНОЮ ТЕРТЯ СМЦ-2

3.1. Вибір технологій розробки

Метою функціонування програмної частини АСУ па ПК є:

– прийняття параметрів функціонування машини тертя СМЦ-2 від оператора;

– прийняття операцій, що мають бути виконані, від оператора та трансформація їх у вигляд, зрозумілий для апаратної частини;

– прийняття показань датчиків (можлива їх обробка);

– відображення показань датчиків.

Для забезпечення цих функцій інтерфейс користувача має бути зручним у користуванні, ознайомленні та швидким у роботі. Існують два різновиди інтерфейсу користувача: інтерфейс командного рядка або *command line interface* (*CLI*) та графічний інтерфейсу користувача або *graphical user interface* (*GUI*).

Інтерфейс командного рядка – різновид текстового інтерфейсу, в якому інструкції ПК передаються тільки введенням з клавіатури текстових команд. Формат виводу інформації в інтерфейсі командного рядка не регламентується; звичайно це простий текстовий вивід, але може бути й графічним, звуковим виводом тощо.

Графічний інтерфейс користувача– тип інтерфейсу, який дає змогу оператору взаємодіяти з ПК через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації. Виконання дій у *CLI* – це безпосередня маніпуляція з графічними елементами.

Зважаючи на порівняння *CLI* та *GUI,* що наведене в порівняльній таблиці 3.1, графічний інтерфейс користувача є більш підходящим для використання у АСУ машиною тертя СМЦ-2, оскільки потребує набагато менше зусиль для ознайомлення,

пропонує достатній рівень контролю за середовищем з одночасною можливістю переглядати стан машини тертя.

Таблиця 3.1

Порівняльна таблиця *CLI* та *GUI*

Параметр	*CLI*	*GUI*
Зручність	Ознайомлення з інтерфейсом командного рядка складніше через необхідність запам’ятовування необхідних для роботи та навігації команд.	Графічний інтерфейс є візуально інтуїтивно зрозумілим, користувачі, як правило, навчаються користуватися графічним інтерфейсом швидше, порівняно з *CLI*.
Контроль	Користувачі мають високу ступінь контролю над файлами та операційними системами в інтерфейсі командного рядка.	Графічний інтерфейс пропонує широкий доступ до файлів, функцій програмного забезпечення та операційної системи в цілому.
Багатозадачність	Багато середовищ командного рядка здатні до багатозадачності, але вони не пропонують однакову легкість і можливість переглядати кілька речей одночасно на одному екрані.	Користувачі графічного інтерфейсу мають вікна, які дозволяють користувачеві одночасно переглядати, контролювати, маніпулювати та перемикати кілька функцій програми.
Швидкість	Користувачам командного рядка потрібно лише використовувати клавіатуру для навігації по інтерфейсу, що часто призводить до швидшої роботи.	Сучасні графічні інтерфейси швидкі та ефективні, для них потрібна миша, тому для введення потрібно рухати руку від миші до клавіатури. Для багатьох користувачів відведення руки від клавіатури для переміщення вказівника миші відбувається повільніше, ніж під час використання командного інтерфейсу.
Ресурси	Комп’ютер, який використовує лише командний рядок, займає набагато менше системних ресурсів комп’ютера, ніж графічний інтерфейс.	Графічний інтерфейс вимагає більше системних ресурсів через додаткові елементи, які вимагають завантаження.

Продовження таблиці 3.1

Параметр	*CLI*	*GUI*
Сценарії	Інтерфейс командного рядка здебільшого вимагає, щоб користувачі вже знали команди сценаріїв та синтаксис, що ускладнює створення сценаріїв новим користувачам.	Створення сценаріїв за допомогою графічного інтерфейсу стало простішим завдяки програмному забезпеченню, яке дозволяє користувачам писати сценарії, не знаючи всіх команд та синтаксису. Програмне забезпечення для програмування містить посібники та підказки щодо кодування певних функцій та параметрів попереднього перегляду, щоб побачити, чи працює сценарій та як він працює.
Віддалений доступ	Під час доступу до іншого комп'ютера або пристрою через мережу користувач може маніпулювати пристроєм або його файлами за допомогою інтерфейсу командного рядка. Однак, необхідно знати команди для цього.	Віддалений доступ до іншого комп’ютера або сервера можливий у графічному інтерфейсі та зручний для переміщення з невеликим досвідом.
Різноманітність	Команди *CLI* зазвичай не змінюються, без крайньої необхідності. Хоча можуть бути введені нові команди, оригінальні команди часто залишаються незмінними.	Кожен графічний інтерфейс має різний дизайн та структуру, коли мова йде про виконання різних завдань. Навіть різні ітерації одного графічного інтерфейсу, такі як *Windows*, можуть мати сотні різних змін між кожною версією.

Не всі мови програмування підходять для розробки графічного інтерфейсу користувача, а серед тих що підходять необхідно обрати мову, що забезпечує високу швидкість розробки та швидкодії інтерфейсу. Серед мов, що найкраще підходять до розробки графічного інтерфейсу знаходяться *C++*, *Java*, *Python*, *VB.net*/*C#*.

Мова *C++* цікава тим, що зазвичай є мовою для розробки рендерингу (графічної візуалізації). Однак це не тому, що її легко використовувати у середовищах розробки, а тому, що вона компілюється. Компіляція в рідний машинний код, як правило, означає, що вона перевершують більшість інших мов за швидкістю роботи програми, що дуже важливо, якщо програма (наприклад, гра) вимагає високих частот оновлення кадрів.

Хоча для *C++* існують крос-платформні бібліотеки графічного інтерфейсу, вони не найпростіші у використанні через їх складну природу порівняно з іншими мовами. Тому *C++* використовується для розробки графічного інтерфейсу лише, якщо швидкість роботи є критичною. Однак, якщо необхідно використовувати лише кілька кнопок, поля редагування та списки, не варто розглядати *C++* для розробки.

Оскільки одним із основних напрямків роботи *Java* є крос-платформенність, доступні пакети графічного інтерфейсу працюють на більшості пристроїв із підтримкою *Java*. Два основних пакети, доступні для програм графічного інтерфейсу *Java, – AWT* (є застарілим) та *Swing*. Однією чудовою особливістю використання *IDE,* як *Eclipse*, є те, що ви можете графічно розробляти графічний додаток (майже ідентично до платформи *.net*), що може заощадити величезну кількість часу на етапі проектування.

Java також має бібліотеки, доступні для графічних процедур, що дозволяє здійснювати як 2*D*, так і 3*D* візуалізацію графіки (одним з популярних прикладів є *Minecraft*, це 3*D*-рендерированная воксельна гра, написана на *Java*). Однак одним недоліком *Java* як графічного середовища кодування є її швидкість, саме тому в *Java* дуже мало написаних 3*D*-ігор. Оскільки *Minecraft* написаний на *Java*, він погано працює на комп'ютерах низького класу та на комп'ютерах середнього класу, доступних для більшості домашніх користувачів.

Якщо проект повинен бути кроссплатформенним і потрібна помірна обчислювальна потужність, *Java* буде гарним вибором мови.

Python цікавий тим, що позиціонує себе як просту у використанні та читанні мову, але створення програми з графічним інтерфейсом може бути складним. *Python* – це інтерпретована мова, і вся вона написана в коді (на відміну від графічного

інтерфейсу *Java* або графічного інтерфейсу *.net*), тому створення програм буде складнішим, а використання такої бібліотеки графічного інтерфейсу, як *wxPython*, не дуже полегшує стан речей.

Проте, *Python* все ще набагато простіший у використанні, ніж *C++*, оскільки імена бібліотек є більш читабельними, потрібно менше рядків коду, і програми на *Python*, зазвичай, містять менше помилок. Але оскільки *Python* – мова, що інтерпретується, швидкість обробки є одною з найгірших. Тому він не підходить для програм, яким може знадобитися обробка великих обсягів даних під час оновлення графічного вікна.

З усіх мов, показаних у цій статті, *VB.net* та *C#* спільно з *Visual Studio* є найпростішими мовами для використання з графічними процедурами. *Visual Studio* надає редактор, який дозволяє створювати програми-форми шляхом графічного їх проектування, що значно зменшує час, необхідний для створення програми з графічним інтерфейсом.

Однак *VB.net* і *C#* страждають від одного суттєвого недоліку - вони не є крос-платформними (лише для *Windows*). Обидві мови покладаються на фреймворк *.net* для створення графічних додатків, недоступних на *Mac*, *Linux* або *Android*. Однак існують деякі бібліотеки, які дозволяють використовувати крос-платформні програми, такі як Mono, але в результаті вони мають менше функціональних можливостей, ніж у разі використання *Windows*.

Оскільки ОС *Windows* – цільова платформа для створення АСУ машини тертя СМЦ-2, то *VB.net* та *C#* є найкращим вибором. Мова *C#* є набагато новішою та розвиненішою ніж *VB.net* тому варто використовувати її.

C# – загально об’єктно-орієнтована мова програмування для розробки взаємодії з мережею та створення веб-загальний. *C#* визначається в межах спільної мовної інфраструктури (*Common Intermediate Language, CLI*). Особливості мови:

– можливість створення властивостей (*properties*), які слугують в якості додатків для приватних змінних членів;

– атрибути, які надають декларативні метадані про типи під час виконання;

– інкапсульовані сигнатури методів, що називаються делегатами, які дозволяють типобезпечні оповіщення про події;

– інтегровані в мову запити до бази даних (*Language Integrated Query, LINQ*), що надають вбудовані можливості, щодо створення запитів до різних джерел даних на уніфікованій мові;

– методи розширення – статично визначений метод, який застосовується до певного типу і може бути викликаний у вихідному коді, так, наче він є членом цього типу;

– велика кількість синтаксичних конструкцій, так званого «синтаксичного цукру», які спрощують визначення певних синтаксичних конструкцій та допомагають зберігати код чистим.

У випадку використання *C#* на *Windows* найкращим, найзручнішим середовищем розробки є *Visual Studio IDE* використання якого надзвичайно сильно спрощує розробку графічних інтерфейсів. Інтегроване середовище розробки *Visual Studio* є креативною стартовою панеллю, яку можна використовувати для редагування, налагодження та створення коду. Інтегроване середовище розробки або *integrated development environment* (*IDE*) – це багатофункціональна програма, яка може використовуватися для багатьох аспектів розробки програмного забезпечення. Крім стандартного редактора та налагоджувача, які постачаються з більшістю *IDE*, *Visual Studio* включає компілятори, засоби завершення коду, графічні дизайнери та багато інших функцій для полегшення процесу розробки програмного забезпечення.

Для *C#* розроблено дві бібліотеки для розробки графічного інтерфейсу: *Windows Forms (WinForms)* та *Windows Presentation Foundation (WPF)*. У *WPF* є новішим засобом створення застосунків та має дуже багато переваг над *WinForms*. *WPF* полегшує написання правильно структурованих програм, у *WinForms* це вимагає зусиль і ручної роботи. Але наслідком і зворотньою стороною цього є набагато більша складність *WPF* як засобу створення графічного інтерфейсу, що сильно збільшує поріг входження для нових користувачів і виомагає знання правильних методик програмування на ньому. Використання *WinForms* буде достатнім для створення графічного інтерфейсу оператора та керуючої логіки АСУ машини тертя СМЦ-2.

Для програмування апаратної частини АСУ необхідно застосовувати засоби, що надаються розробником мікроконтролера. На платі *LPC1768* використовується мікропроцесор *ARM Cortex-M3* для якого розроблено платформа розробки *mbed*. Особливістю *mbed* платформи є середовище розробки і компілятор *ARM C++*. Обидва цих інструменти є повністю безкоштовними, мають веб-інтерфейс і працюють в режимі онлайн, завдяки чому не потрібно завантажувати і встановлювати будь-яке ПЗ. Підтримувані браузери – *Internet Explorer*, *Firefox, Safari, Chrome* на будь-якій операційній системі – *Windows, Mac, Linux*. Це надає змогу зайти в середовище розробки звідки завгодно і продовжити роботу з того місця, на якому зупинилися. Крім того, веб-інтерфейс дозволяє легко ділитись своїм кодом і вести спільну розробку. *ARM* компілятор використовує генерує чистий, ефективний і оптимізований код, який можна безкоштовно використовувати навіть при виробництві. Існуючі *ARM*-програми та заготовки можуть бути перенести на мікроконтролер *LPC1768*.

Однією з переваг цієї платформи є *mbed*-бібліотека, яка представляє собою спеціальний *API* для програмування плати. Вона звільняє розробника від необхідності писати багато низькорівневого коду, як це зазвичай буває при програмуванні мікроконтролерів. Бібліотека надає розробнику *API*-функції і інтуїтивно зрозумілі абстракції, що описують ту чи іншу периферію, які добре налагоджені і дають можливість експериментувати, не замислюючись про внутрішній устрій контролера або його периферійних пристроїв.

Для прошивки *mbed*-платформи не потрібно використовувати зовнішній програматор. При підключенні до комп'ютера через *USB*, платформа визначається в операційній системі як звичайний пристрій. Тому для прошивки досить просто скопіювати на нього скомпільований бінарний файл.

Замість того, щоб просто надавати приклади, mbed фокусується на багаторазовій функціональності бібліотеки, з чіткими інтерфейсами та надійними реалізаціями. Основна бібліотека mbed підтримує основну периферію *LPC1768*, а бібліотеки, які вже надані спільнотою дизайнерів mbed, включають підтримку *USB*, *TCP/IP* та *HTTP*. Також можна додати сторонні стеки та стеки з відкритим кодом.

3.2. Опис апаратної частини модернізованої автоматизованої системи керування та збору інформації

Центром апаратної складової автоматизованої системи управління і обробки інформації є апаратно-обчислювальна платформа *mbed LPC1768*, розроблений для створення прототипів усіх видів пристроїв, особливо тих, що включають *Ethernet*, *USB*, а також безлічі периферійних інтерфейсів та флеш-пам'яті.

Мікроконтролер *mbed NXP LPC1768*, з 32-бітовим ядром *ARM Cortex-M3*, що працює на частоті 96 *МГц*. Він включає 512 *КБ* флеш-пам’ять, 32 *КБ* ОЗУ і безліч інтерфейсів, включаючи вбудований *Ethernet*, *USB*-хост, аналого-цифровий перетворювач (АЦП), цифро-аналоговий перетворювач (ЦАП), широтно-імпульсна модуляція (ШІМ) та інші інтерфейси вводу-виводу. На рисунку 3.1 зображено часто використовувані інтерфейси та їх розташування. Всі пронумеровані контакти (*p*5-*p*30) також можуть використовуватися як цифрові інтерфейси входу/виходу. *NXP LPC1768* підтримує з’єднання з великою кількістю периферійних пристроїв, як у випадку АСУ СМЦ-2.

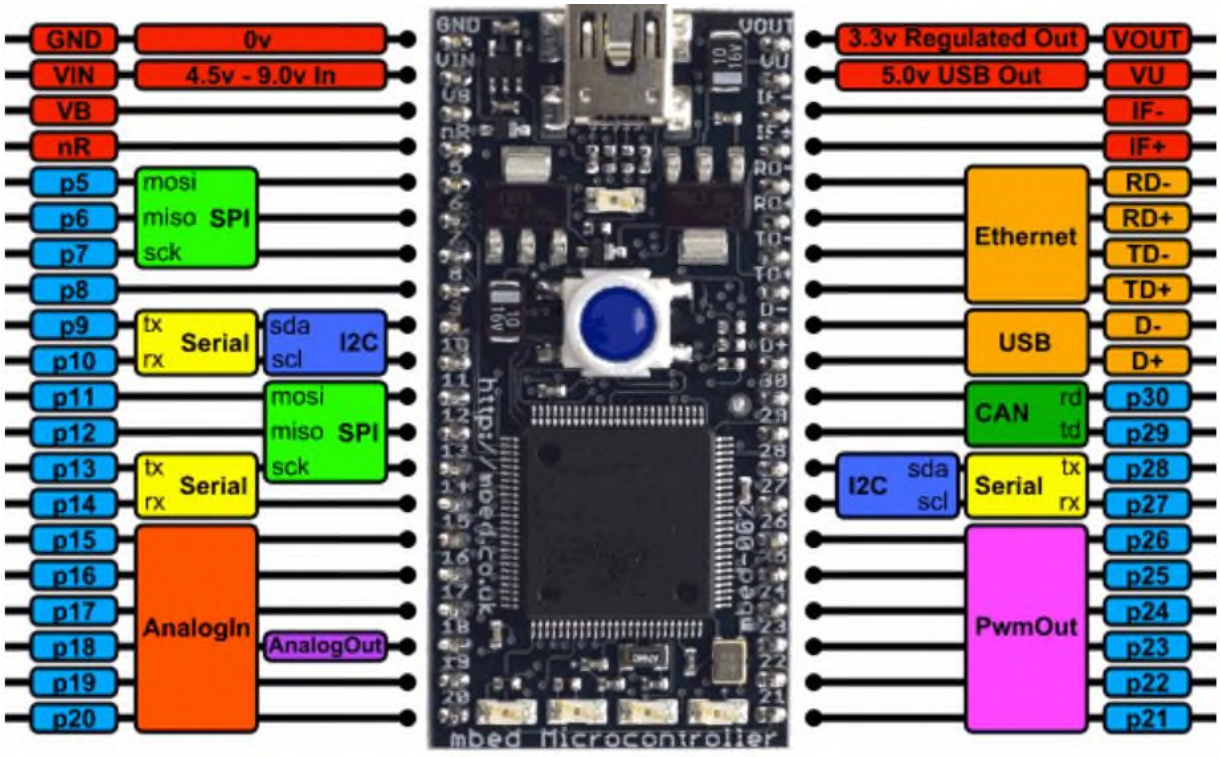

Рис. 3.1. Інтерфейси *NXP LPC1768*

Основні особливості плати *NXP LPC1768*:

– плата підключається до ПК за допомогою *USB*-кабелю, який дозволяє програмувати контролер, живить схему та деякі підключені переферійні пристрої та діє як послідовний порт для комунікації апаратної частини АСУ та ПК;

– живлення може відбуватись від адаптера змінного струму від 4.5 *В* до 9 *В*;

– уже продуктивний та енергоекономний контролер *ARM Cortex-M*3;

– через велику поширеність існує широка програмна база у вигляді високорівневих бібліотек, в мережі викладено багато прикладів коду та проєктів, а також відповідей на різноманітні запитання на форумах;

– *LPC1768* має 26 контактів цифрового та 6 аналогових контактів вводу/виводу;

– *LPC1768* має вбудований функціонал для управління мотором за допомогою ШІМ;

– до *LPC1768* можна під'єднати багато апаратних засобів, що реалізують різноманітні функції від управління мотором до модулів *Wi-Fi* та *Bluetooth*.

Враховуючі всі особливості цієї платформи і повну сумісність всіх необхідних, для реалізації розробленої структурної схеми, додаткових пристроїв з цією платформою, можна зробити висновок, що вона абсолютно підходить для реалізації розробленої моделі автоматизованої системи управління та збору інформації.

Інша частина апаратного забезпечення – це все те обладнання, що контролює та знімає показання роботи СМЦ-2. Для зняття показників температури зразків тертя та рідини будуть використовуватись дві термопари *DS18B20* та пірометр *MLX90614*.

Температурні показники навколишнього та змащувального середовищ реєструються термопарами *DS18B20*, як і до модернізації. Інформація з термопар поступає на мікроконтролер, де вона оброблюється, переводиться у формат градусів Цельсію та передається на ПК. Температура в зоні тертя реєструється пірометром *MLX90614*, який знаходиться в близькій відстані від поверхонь фрикційного контакту. Модуль навантаження та датчик моменту залишаються

незмінними. На зміну тензодатчику *Z6* прийшов тензодатчик *HX711*. *HX711* є аналоговим датчиком та потребує калібрування, яке виражається в підборі коефіцієнту калібрування. Для вимірювання швидкості використовується датчик *LM393*. Для керування двигуном використовується схема з датчиком Хола, як датчика швидкості та оптосимістором, для задання напруги. Всі нові пристрої є дешевшими старих, деякі, навіть, точніші.

Важливим є використання оптосимістора (*MOC*3022*)* для контролювання обертів двигуна П32УХЛ4. Підключення оптосимістора до мікроконтролера стає дуже легкою задачею, завдяки тому, що *NXP LPC1768* має контакти для задання ШІМ сигналу. Оптосимістори належать до класу оптронів і забезпечують дуже хорошу гальванічну розв'язку (порядка 7500 *В*) між керуючою ланцюгом і навантаженням, тобто у разі пробою симістора мікроконтролер не вийде з ладу. Оптосимістори складаються з арсенід-гелієвого інфрачервоного світлодіода, з'єднаного за допомогою оптичного каналу двонаправленим кремнієвим перемикачем.

Залишається питання вибору технології передачі даних. *LPC1768* підтримує велику кількість різних протоколів. Найкращими кандидатами є *Wi-Fi, Ethernet* та послідовна передача даних через *communication*-порт (*COM*-порт) або через *USB*-порт. В порівнянні з *USB RS-232* є набагато простішим, з точки зору реалізації, і повільнішим протоколом, а з точки зору користувача – різниця мінімальна, тому *RS-232* виключається зі списку претендентів. *Wi-Fi* та *Ethernet* – пакетні протоколи передачі даних, тому, в конкретному випадку, при їх використані варто створювати з'єднання *transport control protocol* (*TCP)* разом з бінарним протоколом серіалізації даних, що досягається використанням *Apache Thrift* (або іншою технологією опису інтерфейсів та бібліотекою для виклику віддалених процедур), мови опису інтерфейсів міжпроцесової комунікації. За допомогою використання платформи *mbed* це досягається використанням бібліотеки для роботи з *Ethernet* та написанням не більше п'яти рядків програмного коду для ініціалізації з'єднання та відправки повідомлень. Ще одним аргументом в сторону *Ethernet* в порівнянні з *USB* є швидкість передачі даних. Стандарт *USB2.0* підтримує максимальну швидкість

передачі даних приблизно рівну 50 *МБ* за секунду, а безпосередньо *Ethernet* модуль *LPC1768* здатен передавати дані на швидкостях 100 *МБ* за секунду. Також в кадр Ethernet входить 32-бітне поле циклічного надлишкового коду, що є алгоритмом обчислення контрольної суми і призначений для перевірки цілісності даних, значення контрольної суми вираховується автоматично при інкапсуляції даних у кадр і перевіряється автоматично при декапсуляції. Поле циклічного надлишкового коду звільняє програмне забезпечення від перевірки цілісності отриманих даних. Єдиною і дуже вагомою проблемою є те, що використовуючи *Windows* неможливо напряму отримувати *Ethernet*-кадри з драйверу мережевої плати, оскільки компанія-розробник ОС *Windows* – *Microsoft* з міркувань безпеки. Проте для *C#* існують розроблені бібліотеки *SharpPcap* та *Pcap.net*, що використовує бібліотеку дублювання пакетів *Win*10*Pcap*, що використовують свій власний драйвер мережевої плати та звичайні аналізатори трафіку. Використання цих проміжних рівнів обробки *Ethernet*-кадрів буде сповільнювати комунікацію між ПК та мікроконтролером, оскільки необхідно буде постійно копіювати кадр з одного місця в пам’яті в інше, тому прийняте рішення використовувати інтерфейс *USB* як засіб передачі даних, оскільки це спростить реалізацію програмної частини АСУ.

Найголовнішими перевагами такого вибору є: простота передачі даних, достатня швидкість передачі та те, що завдяки тому, що *NXP PLC*1768 може живитись від *USB*, що під’єднано до ПК, може бути використано одне підключення, яке буде служити живленням та середовищем передачі даних між мікроконтролером та ПК.

3.3. Алгоритм роботи автоматизованої системи управління на основі модифікованої структурної схеми

Розробка алгоритму роботи АСУ є необхідним попереднім кроком до його програмної реалізації. Схема алгоритму роботи АСУ зображена на рисунку 3.2.

Необхідною умовою для правильного функціонування модулю системи керування є встановлення транспортного з'єднання автоматизованого робочого місця з апаратно-обчислювальною платформою *LPC1768*. Алгоритм встановлення транспортного з'єднання автоматизованого робочого місця з апаратно-обчислювальною платформою *LPC1768* наведений на рисунку 3.3.

АРМ надає можливість обрання типу керування, ручне або автоматизоване. Автоматизоване керування полягає в використанні плану виконання, що складається з послідовності записів, кожен з яких має тривалість виконання та задає параметри двигуна та інших складових машини тертя СМЦ-2, які мають бути дійсними протягом цього періоду часу. Під час роботи двигуна змінюються показники датчиків температури, тощо, які в свою чергу відображаються на графіку в режимі реального часу. Всі зібрані результати можуть бути збережені в текстовий файл в будь-який зручний момент, навіть по середині процесу виконання досліду. Для комунікації ПК та *LPC1768* необхідно розробити певний набір повідомлень та порядок їх обміну, що називається протоколом комунікації. Алгоритм встановлення транспортного з'єднання ПК та *LPC1768* використовує цей протокол комунікації на етапі ініціалізації з'єднання. Для початку процесу встановлення з'єднання АРМ має визначити *USB*-порт, до якого підключено *LPC1768* та пройти процес ініціалізації з'єднання. Визначення *USB*-порту здійснюється перебором всіх існуючих *USB*-портів, якщо на порті ідентифіковано повідомлення від мікроконтролера то перевіряється працездатність мікроконтролера шляхом надсилання повідомлення та отримання відповіді про успішне з'єднання. Після завершення ініціалізації *LPC1768* має перейти в операційний стан та почати повідомляти ПК показники датчиків температури протягом строго визначеного інтервалу часу, щоб забезпечити режим реального часу. Забезпечувати режим реального часу і необхідну точність надсилання повідомлень допомагають 32-бітні таймери, що розміщені на платі *LPC1768*. З ПК до *LPC1768* можуть надходити команди, визначені протоколом комунікації, які в свою чергу будуть оброблені, переформовані в зрозумілі для певного периферійного пристрою та надіслані на виконання цьому пристрою.

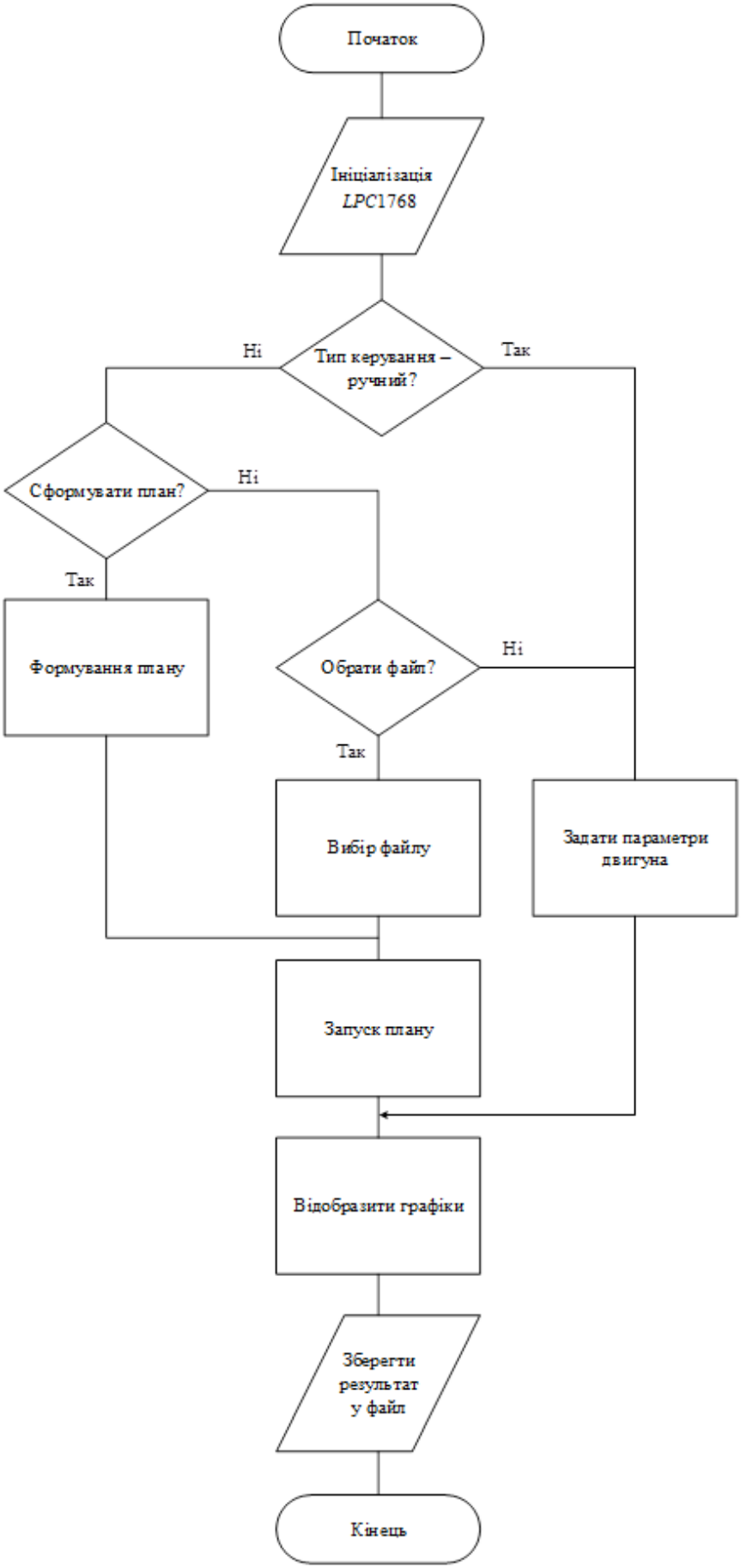

Рис. 3.2. Схема алгоритму роботи АСУ

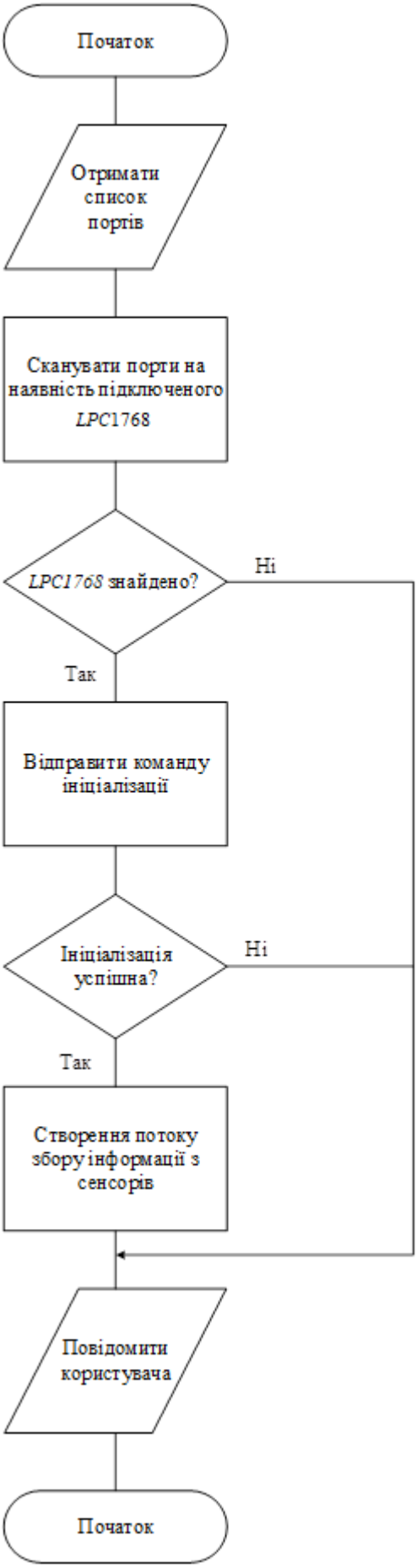

Рис. 3.3. Схема алгоритму встановлення транспортного з’єднання *LPC1768* та АРМ

3.4. Опис процесу розробки графічного інтерфейсу та логіки функціонування програмної частини

Для програмування *LPC1768* використовується мова програмування *C++*, версії 1998 року та середовище програмування *mbed*. Для компіляції коду використовується *ARM* компілятор. Програма для *LPC1768*, як і звичайний *C++* код має містити функцію main, що означає стартову точку входу. В результаті компіляції програми компілятор генерує бінарний файл, готовий до виконання на процесорі з *ARM*-архітектурою.

Однією з найважливіших вимог до користувацького інтерфейсу є реактивність (здатність за будь який обставин відповідати на дії користувача). Це реалізовано за допомогою створення окремих потоків виконання для дій, що можуть блокувати потік виконання на час, помітний для користувача.

Інтерфейс користувача називається «реактивним», якщо він має такі характеристики:

– користувач відразу отримує ефект кожного «жесту»;

– користувач завжди в курсі стану своїх даних.

Інтерфейс користувача реалізується згідно діаграми прецедентів, що зображена на рисунку 2.2. Логічна частина будувалась таким чином, щоб будь-яку дію, яка вимагає великої кількості процесорного часу для виконання, виконувати в потоці, який не є потоком відображення графічного інтерфейсу за допомогою асинхронного підходу до виконання процедур. Згідно діаграми, основними областями взаємодії є: перегляд показників датчиків, меню керування СМЦ-2, виконання плану, перегляд графіків, що відображають значення параметрів. Оскільки область перегляду показників датчиків не є великою за площею та є однією з найнеобхідніших, то прийнято рішення постійно відображати цю область. Для відображення областей графіків, плану, керування використано елемент, що дозволяє відображати одне з трьох вкладень одночасно. Безпосередньо інтерфейс зображено на рисунку 3.4.

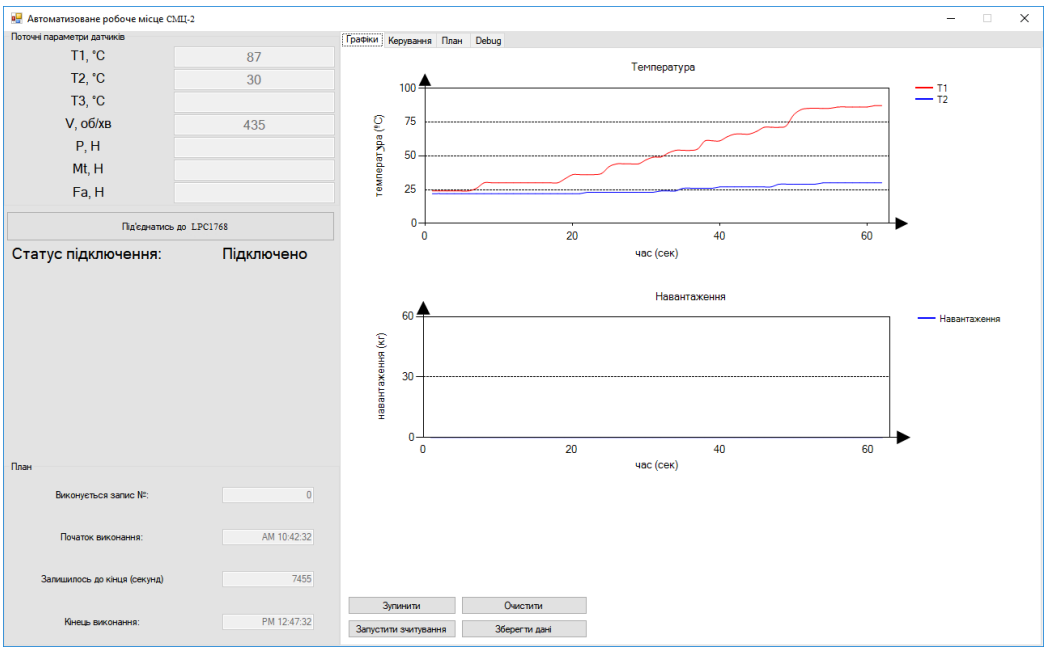

Рис. 3.4. Інтерфейс АСУ, панель «Графіки»

Після запуску зчитування дані з датчиків будуть записуватись в оперативну пам’ять та відображатись на спільному графіку кожну секунду. Зчитування можна зупинити та вивантажити всі дані з оперативного запам’ятовуючого пристрою в файл у текстовому форматі в будь-який момент.

Вкладення «Керування», що зображено на рис. 3.5, надає доступ до керування параметрами СМЦ-2 та відображає встановлені оператором налаштування.

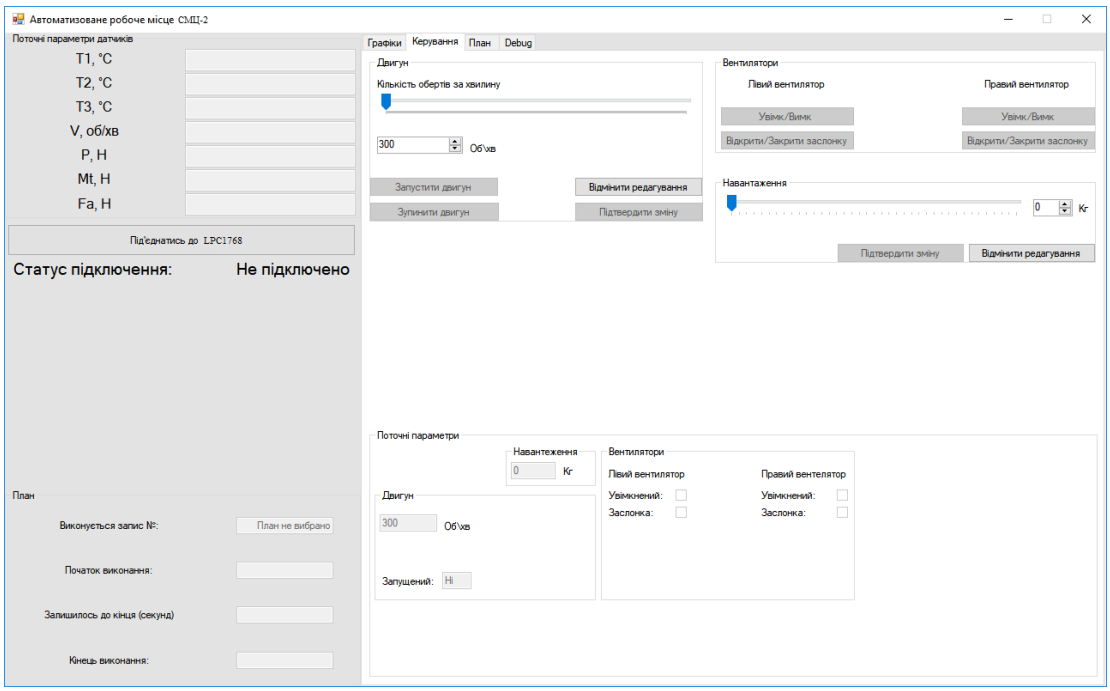

Рис. 3.5. Інтерфейс АСУ, панель «Керування»

Головним інструментом для роботи з СМЦ-2 є не ручне керування, а виконання машиною попередньо розробленого плану робіт. План – це послідовність записів, кожен з яких визначає тривалість виконання та параметри двигуна при проведенні досліджень. Вкладення «План», зображене на рис. 3.6, надає доступ до редагування плану, збереження його в файл та завантаження з файлу. Після запуску плану на виконання все ручне керування блокується, панель «План» в лівому нижньому куті інтерфейсу активується та виводить інформацію про запис, що виконається.

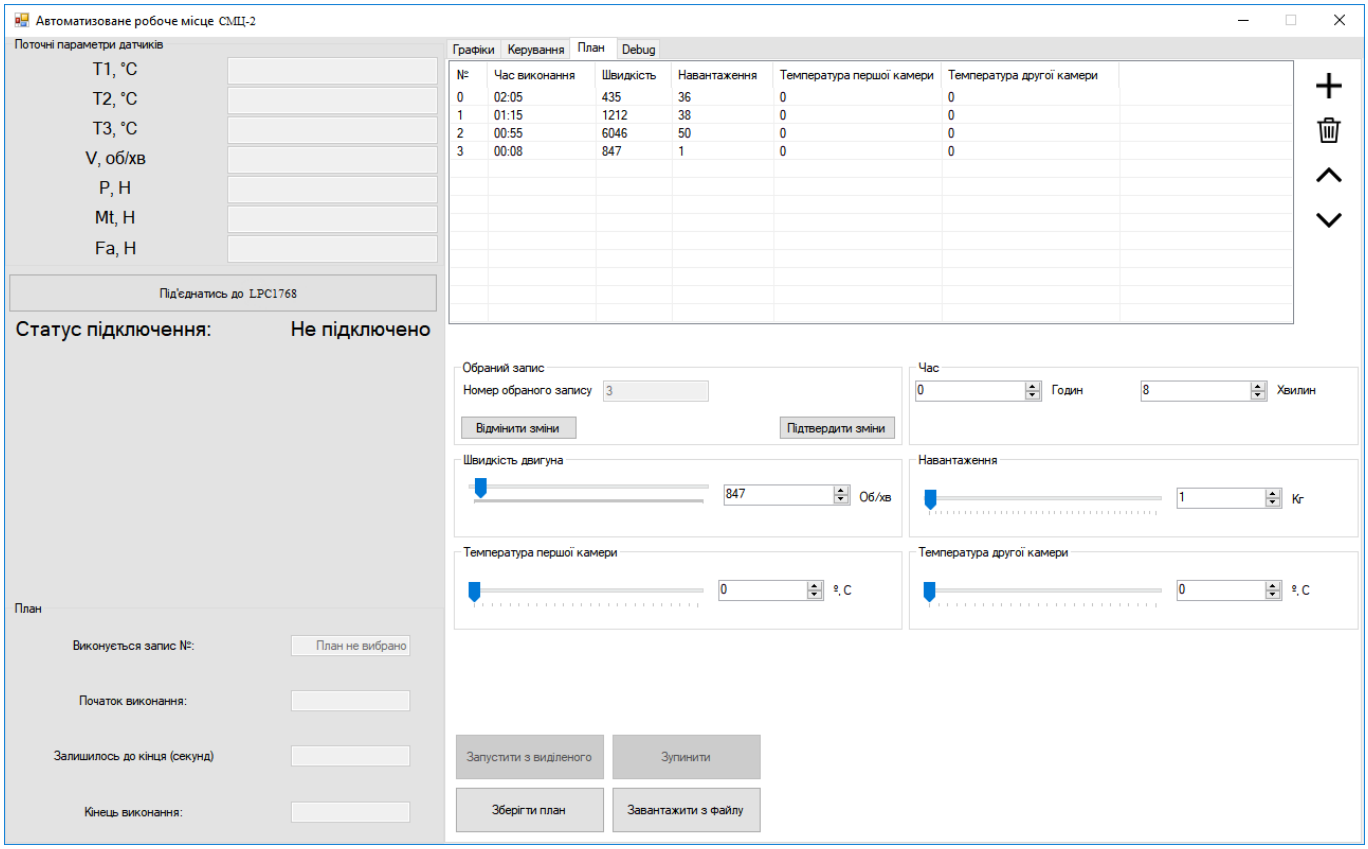

Рис. 3.6. Інтерфейс АСУ, панель «План»

Для роботи мікроконтролера написано програму, наведену в Додатку Б. У функції *setup* оголошуються вхідні та вихідні контакти, а саме вихідні контакти для управління блоком реле, вхідні контакти для зчитувань показників з датчиків температури, вихідний контакт керування оптосимістором, що керує кількістю обертів двигуна. Функція *loop* поділяється на дві частини: зчитування, обробка,

відправка показань сенсорів та прийняття команд оператора і приведення їх у форму, що є зрозумілою для конкретного пристрою, якому ця команда адресована.

3.5. Опис протоколу комунікації апаратної та програмної складових

Інтерфейсом між СМЦ-2 та АРМ виступає мікроконтролер *LPC1768*. Інтерфейс – це набір команд, що будуть зрозумілими з однієї і з іншої сторони. Інтерфейс мікроконтролер-ПК зображений на рисунку 3.7. Завдяки *LPC1768* ПЗ АРМ не має уявлення про конкретне обладнання, яке використовується для отримання інформації, що задовольняє вимогам декомпозиції.

Рис. 3.7. Діаграма послідовностей комунікації через інтерфейс *LCP1768*

Для початку роботи між *LCP1768* та АРМ має відбутись процес ініціалізації. Якщо *LCP1768* не ініціалізоване до роботи і воно підключено до ПК, то відправляються повідомлення запиту ініціалізації. Якщо ПК відправить у відповідь

код початку роботи, то *LCP1768* набуде статусу ініціалізованого та почне зчитувати та відправляти дані сенсорів на ПК. Будь-яка команда оператора відправляється на *LCP1768*, обробляється та в залежності від пристрою, якому вона адресована, формується команда в потрібному форматі і відправляється на відповідний пристрій СМЦ-2.

Для сигналізації про від'єднання від ПК або неполадок у підключення був розроблений механізм повідомлень, що підтримують підключення, а саме ПК кожні 20 секунд відправляє повідомлення на *LCP1768*, сигналізуючи, що підключення досі використовується. Якщо *LCP1768* протягом 45 секунд не отримало таке повідомлення, то контролер переходить у неініціалізований стан та перестає зчитувати дані з сенсорів.

3.6. Висновки до розділу

1. Описано апаратну частину автоматизованої системи керування та збору інформації, обрано технологію передачі даних між *LCP1768* та ПК.

2. Проведено порівняння технологій програмування інтерфейсу користувача. Перевагу надано розробці графічного інтерфейсу на мові *C#* за допомогою бібліотеки *WinForms* у інтегрованому середовищі розробки *Visual Studio*.

3. Розроблено та реалізовано алгоритм роботи АСУ на основі модернізованої структурної схеми автоматизованої системи керування СМЦ-2.

4. Описано процес розробки графічного інтерфейсу та логіки функціонування АСУ керування.

5. Наведено діаграму послідовностей, що описує протокол комунікації апаратної та програмної складових.

РОЗДІЛ 4
ОПИС ПРОЦЕСУ ВИКОРИСТАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ МАШИНОЮ ТЕРТЯ СМЦ-2

4.1. Тестування автоматизованої системи управління

Тестування програмного забезпечення – це процес технічного дослідження, що проводиться з метою виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись:

– відповідність вимогам, якими керувалися розробники;

– відповідність очікуванням для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з програмним забезпеченням та операційними системами;

– відповідність до задач замовника.

Оскільки число можливих тестів навіть для нескладного програмного забезпечення практично нескінченне, то стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявних ресурсів. Як результат програмне забезпечення тестується ручним виконанням програми з метою виявлення баґів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників.

Тестування може проводитись, як тільки створено програмне забезпечення (навіть частково завершено). Процес розробки зазвичай передбачає коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових

програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

У залежності від поставлених цілей види тестування можна розділити на наступні типи: функціональні, нефункціональні, регресійні (пов'язані зі змінами).

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (*component* або *unit testing*), інтеграційному (*integration testing*), системному (*system testing*) і приймальному (*acceptance testing*).

Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

– функціональне тестування (*functional testing*);

– тестування безпеки (*security and access control testing*);

– тестування взаємодії (*interoperability testing*).

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, «як» система працює. Далі перераховані основні види нефункціональних тестів:

– тестування навантаження (*performance and load testing*);

– стресове тестування (*stress testing*);

– тестування стабільності або надійності (*stability* або *reliability testing*);

– об'ємне тестування (*volume testing*);

– тестування встановлення (*installation testing*);

– тестування зручності користування (*usability testing*);

– тестування на відмову і відновлення (*failover and recovery testing*);

– конфігураційне тестування (*configuration testing*).

Після проведення необхідних змін, таких як виправлення дефекту, програмне забезпечення повинне бути перетестоване для підтвердження того факту, що проблема була дійсно вирішена.

В даному дипломному проекті під час розробки використовувалось ручне, функціональне, модульне та системне тестування за тест-кейсами методом білого ящика.

Тестування за тест-кейсами – це тестування за попередньо описаним планом тестування, що оформлюється у тест-кейси. Такий вид тестування застосований з метою формалізації процесу тестування, що в перспективі зменшує загальний час, необхідний для покриття ключових аспектів функціонування програмного модулю.

Ручне тестування – тестування ПЗ без використання додаткових програмних засобів. Ручне тестування використано тому, що воно дозволяє охопити всі функції ПЗ в менш стислі строки, на відміну від автоматичного тестування. Такий вид тестування дозволяє тестувати новий функціонал одразу, без розробки автоматичних тестів; розробка автоматизованих тестів займає багато часу, оскільки потрібно розробити платформу для запуску автоматизованих тестів.

Функціональне тестування – тестування функцій АСУ та коректності їх реалізації.

Модульне тестування – перевірка коректної роботи окремих одиниць системи ПЗ. Цей вид тестування необхідно використовувати одразу після розробки або модифікації певного модулю. В свою чергу системне тестування – це тестування роботи всієї системи. Цей вид тестування необхідно використовувати, щоб підтвердити, що нові зміни не порушують роботу системи в іншому місці.

Тестування методом білого ящика – тестування програмного продукту з доступом до сирцевого коду. Цей вид тестування дозволяє протестувати роботу найбільш вразливих фрагментів програми.

Нижче перелічено розроблені тест-кейси та результати заключного тестування за ними. Всі тест-кейси поділені на дві групи, за класифікацією позитивності сценарію, а саме позитивні та негативні.

Позитивне тестування полягає в перевірці програмного продукту на відповідність очікуваній поведінці. Головною метою такого тестування є перевірка коректності роботи програми. Апаратно-програмний модуль перевірено за наступними тест-кейсами:

– створити новий план, додавати, редагувати, видаляти записи, зберегти план у файл, перевірити вміст файлу;

– завантажити план з файлу, запустити план на виконання, відобразити графіки, зберегти показання датчиків температури у файл, перевірити вміст файлу;

– використовуючи ручне керування запустити двигун, запустити зчитування показників, змінювати швидкість роботи двигуна, зупинити двигун, зупинити зчитування значень, зберегти показання датчиків температури у файл.

Всі тести позитивного тестування були пройдені без зауважень до роботи програми, отже при коректній експлуатації та при відсутності непередбачуваних ситуацій апаратно-програмний модуль поводить себе коректно.

Негативне тестування перевіряє, чи буде програмний продукт працювати в разі, коли поведінка користувача або зовнішніх факторів відрізняється від очікуваної. Модуль перевірено за наступними негативними тест-кейсами:

– відключити *LPC1768* від ПК, запустити автоматизоване робоче місце, підключити *LPC1768* після виводу помилки про підключення, натиснути кнопку «Підключитись до *LPC1768*», перевірити коректність роботи;

– обрати план виконання, запустити його на виконання, у процесі виконання плану відключити *LPC1768* від ПК. У випадку, якщо *LPC1768* не отримує повідомлення про продовження з’єднання від ПК протягом 45 секунд, то відбувається зупинка двигуна, після чого *LPC1768* переходить у початковий стан;

– створити файл, що містить набір випадкових символів, вибрати цей файл при завантаженні плану. В результаті виконання цього тест-кейсу виведено повідомлення, що формат плану не є коректним та наведено формат правильного плану.

В результаті виконання всіх негативних тест-кейсів апаратно-програмний модуль правильно обробляв всі ситуації.

Під час виконання заключного тестування дефектів не виявлено, програма правильно реалізує всі закладені функції та оброблює неочікувані події.

4.2. Інструкція користувача

Дана довідка розроблена для операторів автоматизованої системи управління машиною тертя СМЦ-2 з метою полегшення роботи з автоматизованим робочим місцем.

Програма повинна забезпечувати:

– встановлення транспортного з'єдання з мікроконтролером системи управління;

– можливість розробки плану виконання випробувань на тертя та зношування, що складається з послідовності записів, кожен з яких має тривалість виконання та задає параметри двигуна та інших складових машини тертя СМЦ-2, які мають бути дійсними протягом цього періоду часу;

– можливість запуску плану виконання з будь-якого запису і до кінця, можливість зупинення випробовування в будь-який момент;

– можливість проведення випробувань за допомогою ручного керування параметрами пристроїв апаратної складової системи;

– можливість перегляду налагоджувальної інформації (режим відображення усіх показників системи або режим відображення інформації, що прийнята з мікроконтролера).

При запуску АРМ відбувається ідентифікація *COM*-порту до якого підключено мікроконтролер шляхом сканування усіх наявних *COM*-портів. Приклад повідомлення оператора про сканування *COM*-порту наведений на рисунку 4.1.

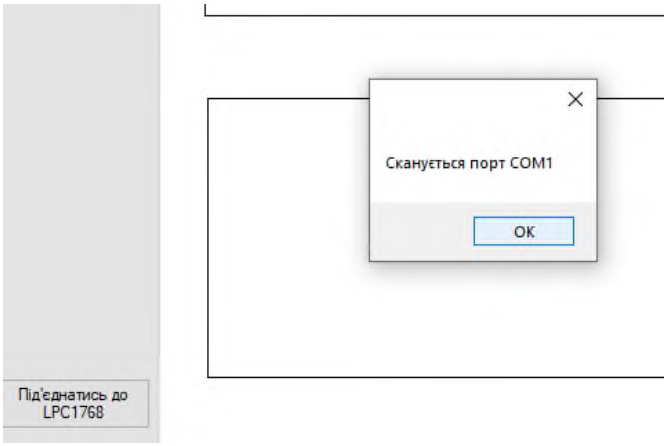

Рис. 4.1. Приклад повідомлення про сканування *COM*-порту

У випадку, якщо мікроконтролер не був виявлений на жодному з *COM*-портів буде виведено повідомлення-помилка (рис. 4.2), що свідчить про те, що необхідно перевірити стан *USB* з'єдання АРМ з мікроконтролером.

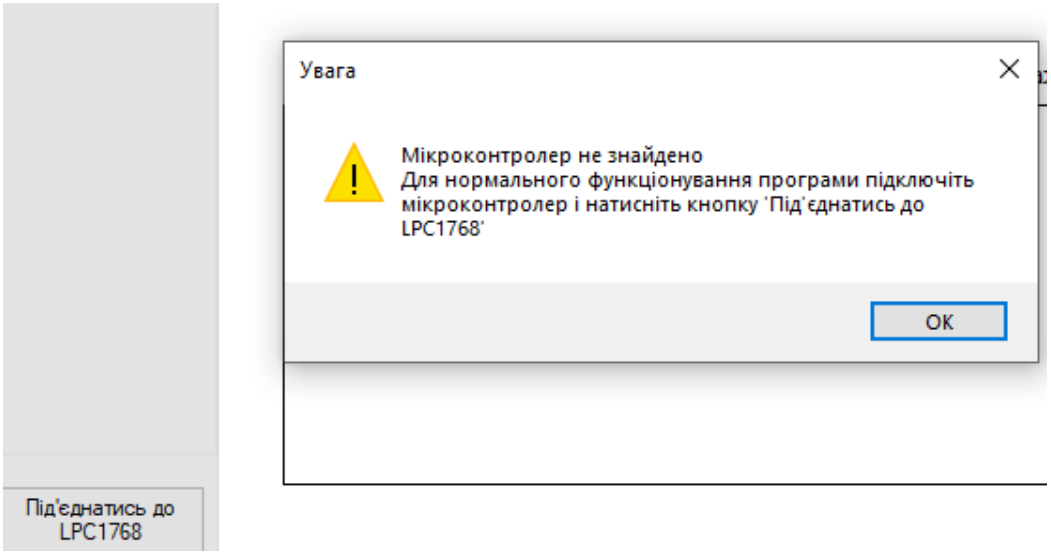

Рис. 4.2. Повідомлення про відсутність ознак роботи мікроконтролера

У разі, якщо мікроконтролер виявлений на одному з *COM*-портів, про це буде повідомлено оператора автоматизованого робочого місця повідомленням, що зображено на рисунку 4.3.

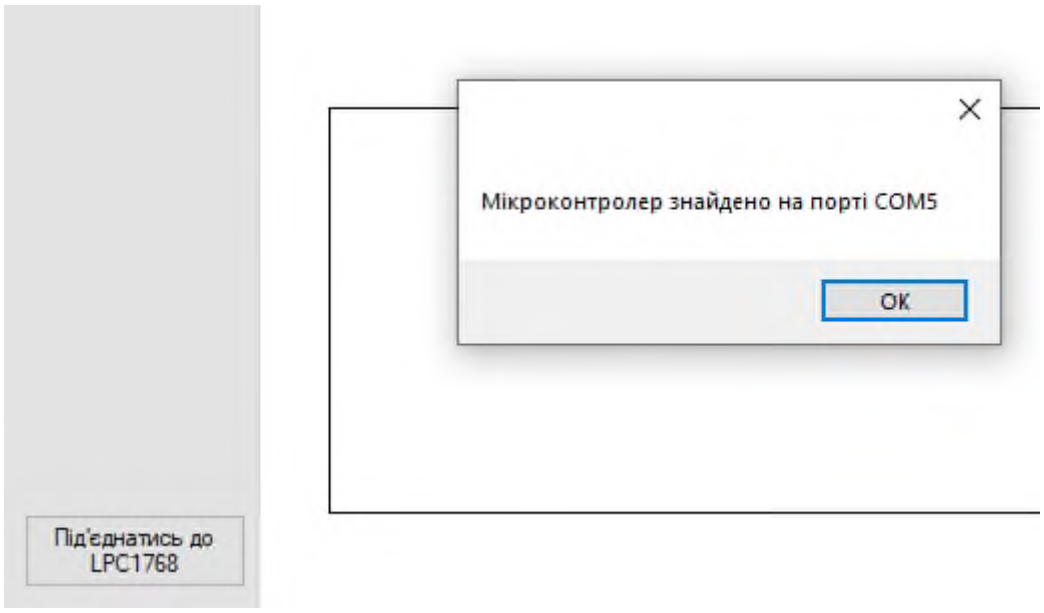

Рис. 4.3. Повідомлення про успішне створення транспортного з'днання

У разі порушення протоколу комунікації однією зі сторін обмін повідомленнями: показниками датчиків та командами буде заборонено. Якщо протокол спілкування порушений зі сторони *LPC1768* оператору буде виведено повідомлення, що зображено на рисунку 4.4. Порушення протоколу комунікації свідчить про внутрішню логічну помилку у програмі АРМ або мікроконтролера.

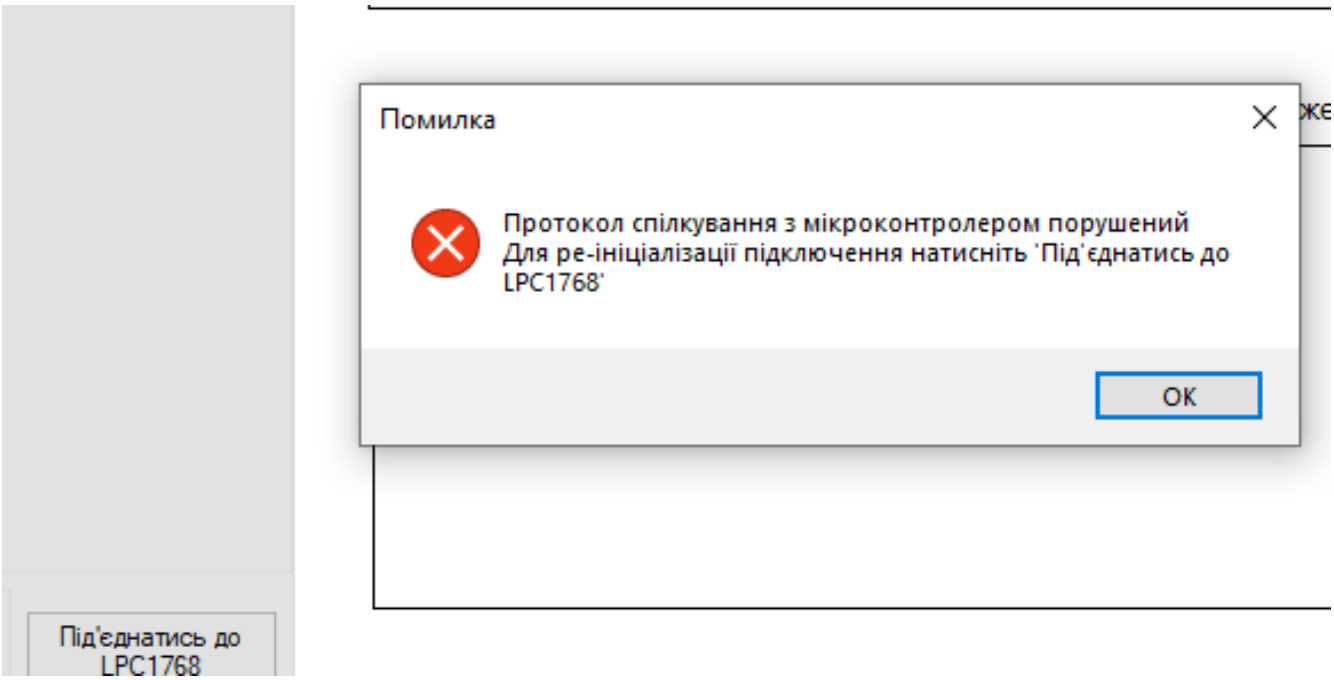

Рис. 4.4. Повідомлення щодо порушення протоколу комунікації зі сторони мікроконтролера

Основне вікно АРМ поділено на 2 вертикальні частини: ліворуч знаходяться поточні параметри датчиків, що оновлюються кожну секунду, стан підключення до мікроконтролера, поточний запис плану на виконанні, що знаходиться на виконанні та час, що залишився до кінця його виконання; праворуч знаходиться меню, що містить чотири вкладення.

На вкладенні графіки (рис. 4.5) розміщені графіки підключених датчиків (наразі температури та навантаження), що оновлюються кожну секунду, також панель керування ними (запуск, зупинка, очищення та вивантаження показників в файл).

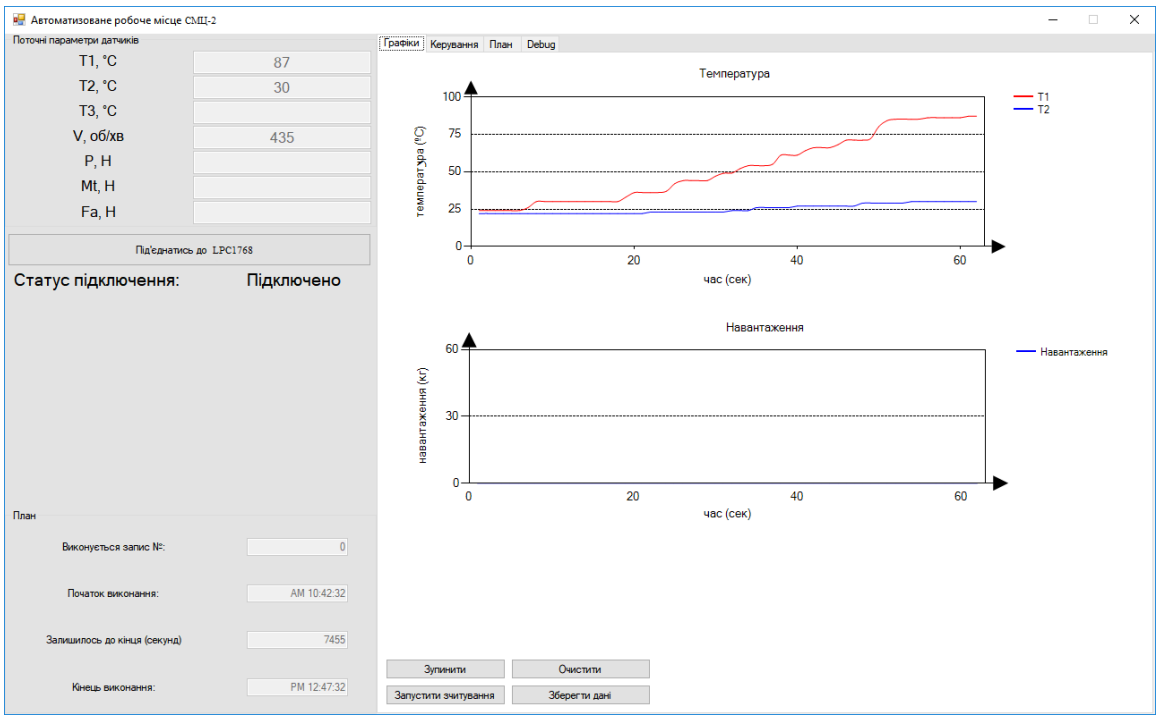

Рис. 4.5. Вкладення «Графіки»

На вкладенні керування (рис. 4.6) розміщенні панель ручного контролю периферійними пристроями (двигуном, вентиляторами, навантаження) та панель поточного стану цих пристроїв.

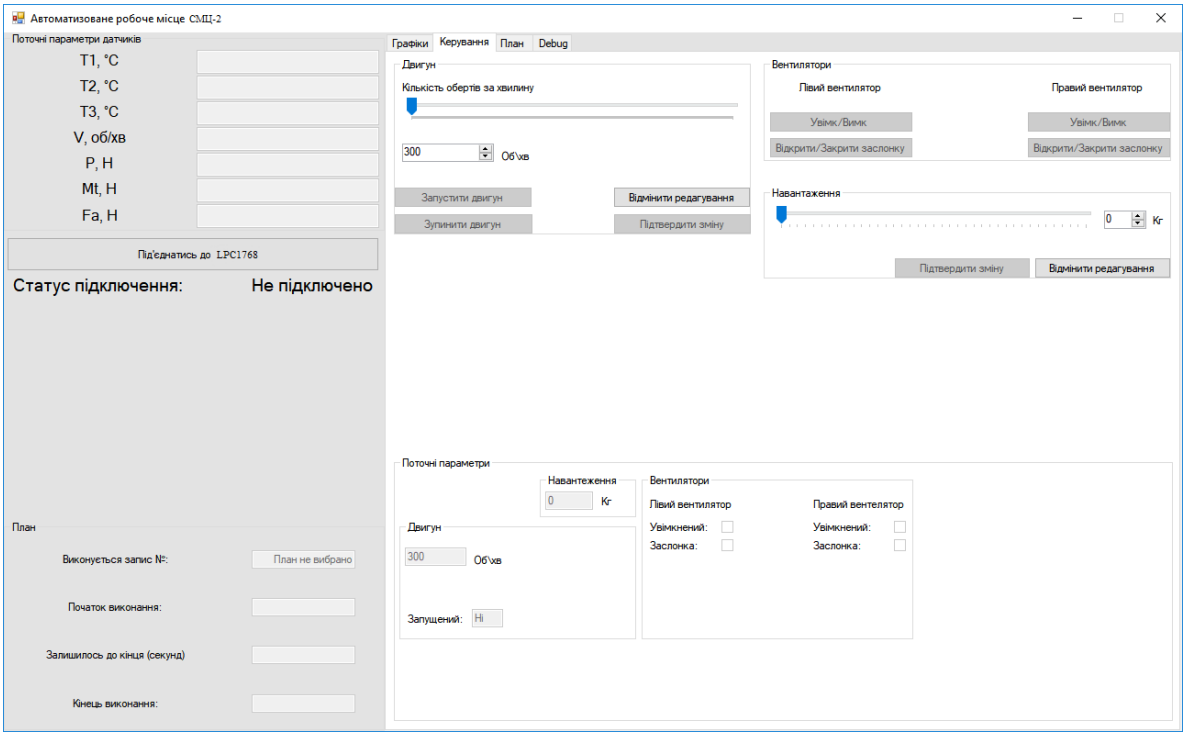

Рис. 4.6. Вкладення «Керування»

На вкладенні «План» (рис. 4.7) розміщений засіб створення плану виконання досліджень на машині тертя СМЦ-2 та кнопки запуску та запуску з виділеного запису. Підтримуються функції вивантаження плану в файл та завантаження плану з файлу.

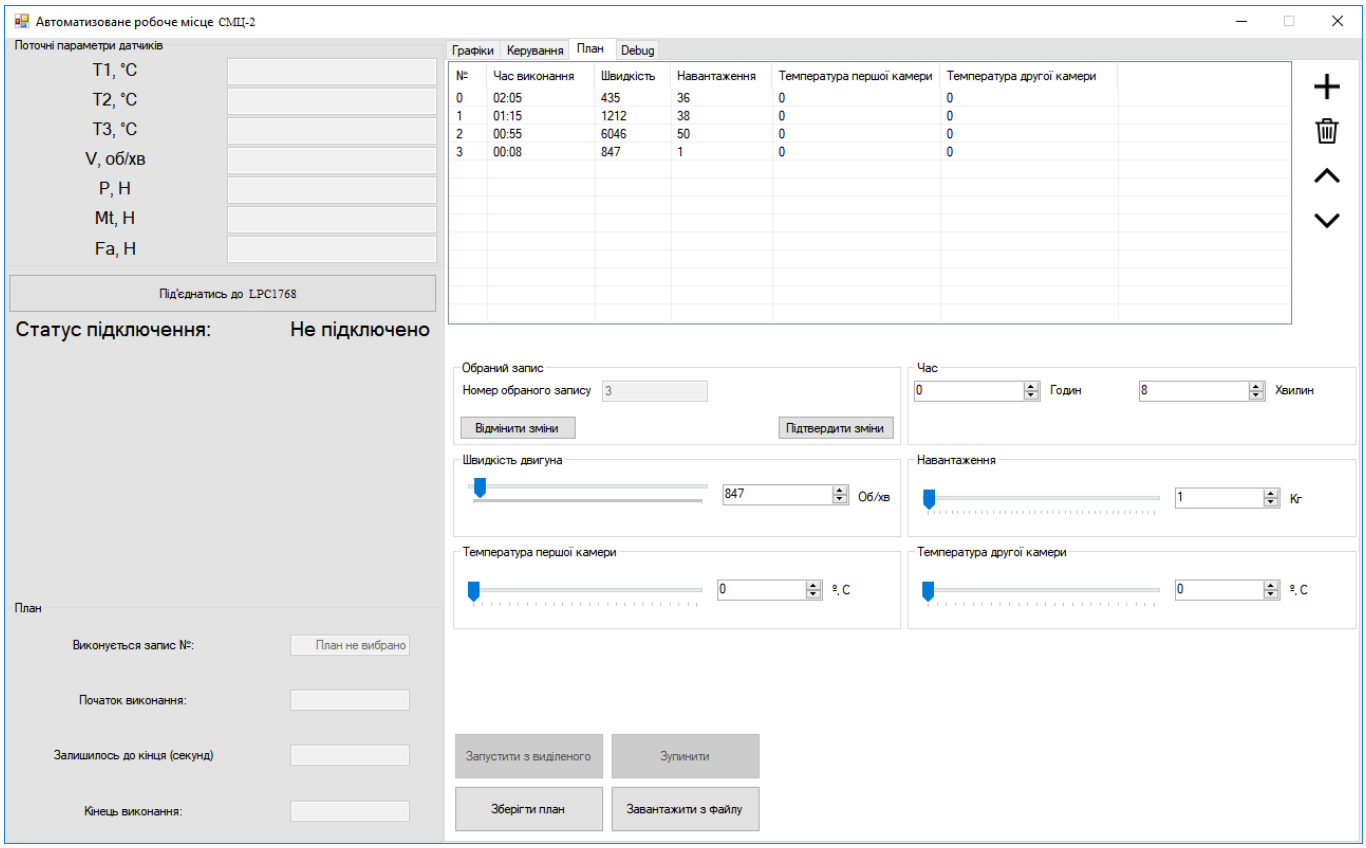

Рис. 4.7. Вкладення «План»

На вкладенні «*Debug*» (відладка) розміщений засіб відладки стану АСУ шляхом виводу всієї доступної інформації про стан системи або повідомлень, що пересилаються через транспортний канал від мікроконтролера до АРМ. Відлагоджувальна інформація, що отримується з вкладення «*Debug*» може бути передана розробнику з метою визначення місця в якому порушена логіка роботи програми та умови за яких неправильна логіка виконується. Приклад вкладення «*Debug*» наведений на рисунку 4.8.

Таким чином в підрозділі наведена інструкція користувача, яка є основною довідкою системи та навчання всіх користувачів автоматизованої системи керування машиною тертя СМЦ-2.

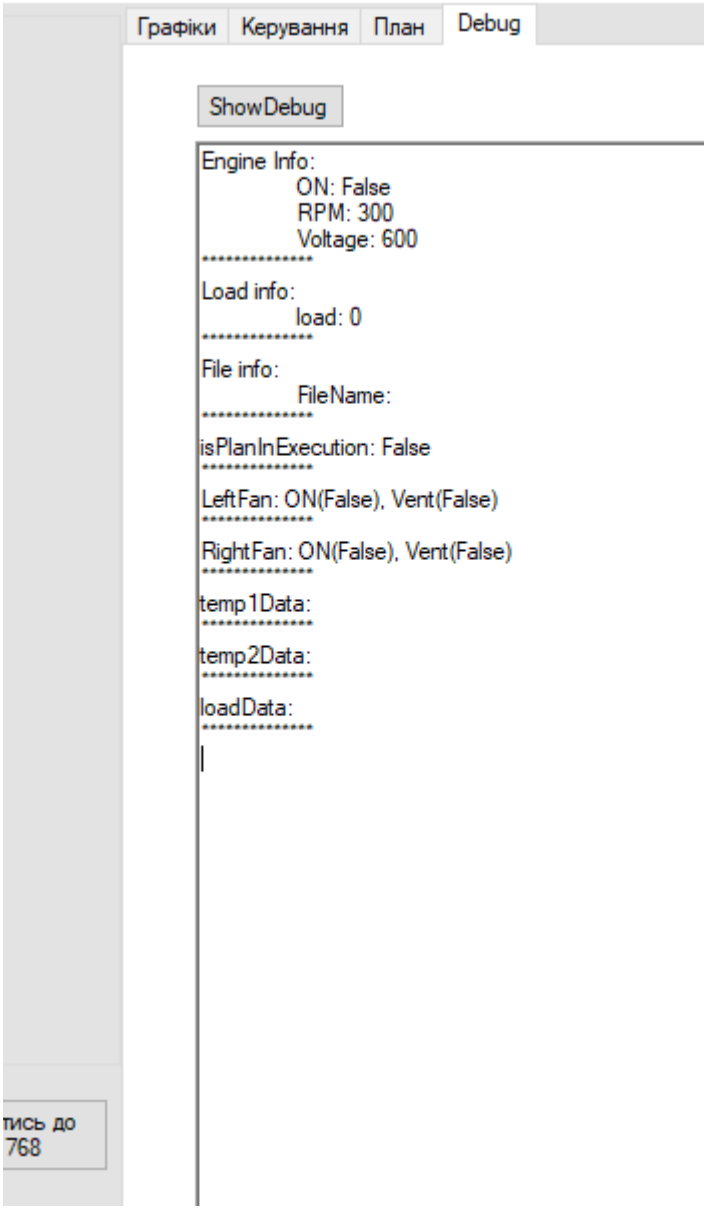

Рисунок 4.8. Вкладення «*Debug*»

4.3. Висновки до розділу

1. Проведено заключне тестування апаратно-програмного комплексу керування.

2. Наведено коротку інструкцію-огляд з експлуатації автоматизованого робочого місця АСУ машини тертя СМЦ-2.

ВИСНОВКИ

1. Розглянуто основні принципи функціонування та класифікацію автоматизованих систем керування, переваги автоматизації систем керування. Досліджено класифікацію приладів контролю та автоматичного регулювання.

2. Описано реологічні властивості реологічних масел, щодо яких проводяться дослідження на машині тертя СМЦ-2.

3. Досліджено процес проведення випробувань на тертя та зношування.

4. Описано спеціалізовану машину тертя СМЦ-2, її автоматизовану систему керування, встановлено її недоліки.

5. Поставлено завдання проектування.

6. Проаналізовано технології та методи проектування автоматизованих комп'ютерних систем управління. Описано принципи побудови, що дозволяють знизити вартість та скоротити час розробки. Для розробки автоматизованої системи керування обрано комбінований підхід з використанням переваг програмного та апаратного способу.

7. Проведено порівняння представників сучасної елементної бази на прикладі мікроконтролерів на платі та одноплатних комп'ютерів. Серед мікроконтролерів було виконано порівняння *Arduino Uno, mbed NXP LPC1768, Intel Galileo* в результаті чого обрано мікроконтролер *LPC1768,* для реалізації централізованого керування периферією.

8. Розроблено модернізовану структуру автоматизованої системи керування СМЦ-2 враховуючи використання платформи *NXP LPC1768,* як інтерфейсу між периферією та ПК. Створено структурну схему модернізованої структури автоматизованої системи управління.

9. Розроблено логічну схему у вигляді діаграми прецедентів мови моделювання *UML*.

10. Описано апаратну частину автоматизованої системи керування та збору інформації, обрано технологію передачі даних між мікроконтролером *LPC1768* та автоматизованим робочим місцем на ПК.

11. Проведено порівняльний аналіз різновидів інтерфейсу, підходів і технологій, що дозволяють проводити розробку інтерфейсу автоматизованої системи управління. Було виявлено, що доцільним є розробка графічного інтерфейсу користувача мовою *C#* за допомогою технології *Windows Forms* в інтегрованому середовищі розробки *Visual Studio*. Головною причиною використання мови *C#* була вимога підтримки операційної системи *Microsoft Windows*, для якої необхідно розробити автоматизовану систему управління машиною тертя СМЦ-2.

12. Описано апаратну частину модернізованої автоматизованої системи управління СМЦ-2, перелічено її переваги.

13. Розроблено та реалізовано алгоритм роботи апаратно-програмного модулю на основі модернізованої структурної схеми автоматизованої системи керування СМЦ-2.

14. Описано процес розробки графічного інтерфейсу та логіки функціонування програмної частини апаратно-програмної системи керування машиною тертя.

15. Проведено порівняння засобів транспортного з'єднання, наведено діаграму послідовностей, що описує протокол комунікації апаратної та програмної складових автоматизованої системи керування, що включає процес встановлення логічного з'єднання.

16. Проведено заключне тестування апаратно-програмного комплексу керування.

17. Наведено коротку інструкцію-огляд з експлуатації автоматизованого робочого місця АСУ машини тертя СМЦ-2.

Практична цінність результатів проекту полягає у наданні можливості:

– відновити проведення триботехнічних випробувань на дослідження реологічних властивостей індустріальних мастил, що дозволить отримувати нові наукові знання;

– автоматизувати проведення дослідних випробувань, що дозволить виконувати довготривалі експериментальні дослідження, підвищить їх точність та якість;

– модернізувати елементну базу, що дозволить зменшити вартість, габарити, затрати на енергоживлення та спростить схему керуючого та знімаючого показники обладнання.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Протасов А. Г. Универсальное устройство для сбора данных с аналоговых и цифровых преобразователей / А. Г. Протасов, А. С. Корогод, Е. Ф. Суслов // Вісник Національного технічного університету України "Київський політехнічний інститут". Серія : Приладобудування. – 2015. – Вип. 49. – С. 145-152

2. *DiStefano J. J. Feedback and control systems / J. J. DiStefano, A. R. Stubberud, I. J. Williams., 1967. – 512 p.*

3. Яшкин И. И. Курс теории автоматического управления / И. И. Яшкин. – Москва: Наука, 1986. – 352 с.

4. ДСТУ 2709-94 Державна система забезпечення єдності вимірювань. Автоматизовані системи керування технологічними процесами. Метрологічне забезпечення. Основні положення. - Київ: Держстандарт України, 1994. – 12 с.

5. Максвелл Д. К. Теория автоматического регулирования / Д. К. Максвелл, И. А. Вышнеградский, А. Стодола. – Москва: Издательство Академии Наук СССР, 1949. – 432 с.

6. *Vasishth, A. Study of Rheological Properties of Industrial Lubricants / A. Vasishth,, P. Kuchhal, G. Anand. – 2014. – P. 1–5.*

7. Верещагин В. И. Методы контроля и результаты исследования состояния трансмиссионных и моторных масел при их окислении и триботехнических испытаниях : монография / В. И. Верещагин, В. С. Янович, Б. И. Ковальский. – Красноярск : Сиб. федер. ун-т, 2017. – 208 с.

8. Беркович И.И. Трибология. Физические основы, механика и технические приложения / И.И. Беркович, Д.Г. Громаковский. – Самара: СГТУ, 2000. – 268 с.

9. *Davim J. P. Tribology for engineers: A practical guide / J. P. Davim. – Woodhead publishing, 2011. – 320 p.*

10. Универсальный испытательный комплекс по определению триботехнических характеристик смазочных материалов на базе серийной машины трения СМЦ-2 / М.Харченко, С. Нефедьев, О. Осипова, Р. Дема. // Известия высших учебных заведений. Машиностроение. – 2017. – №10. – С. 60–68.

11. Филоненко. С. Ф. Акустическая эмиссия. Измерение, контроль, диагностика / С.Ф. Филоненко. – Киев: КМУГА, 1999. – 312 с.

12. Сотников А. Л. Внедрение систем диагностирования / А. Л. Сотников, В. А. Сидоров, А. В. Лукичёв // Машинознавство і деталі машин: Матеріали 4-ої регіональної науково-методичної конференції. – 2002. – с. 81–87.

13. Лужнов Ю. М. Основы триботехники / Ю. М. Лужнов, В. Д. Александров. – Москва: МАДИ, 2013. – 136 с.

14. Параллельная обработка информации: в 5-ти т. – Т.5: Проблемно-ориентированные и специализированные средства обработки информации / А. И. Аксенов, В. В. Аристов, Е. Ю. Барзилович и др. – Киев: Наукова думка, 1990. – 504 с.

15. Цмоць І. Г. Інформаційні технології та спеціалізовані засоби обробки сигналів і зображень у реальному часі / І. Г. Цмоць. – Львів: УАД, 2005. – 227 с.

16. Цмоць І. Г. Принципи розробки та оцінка основних характеристик комп'ютерних систем реального часу / І. Г. Цмоць, Я. П. Кісь, В. Р. Іванців // Науково-технічний журнал "Технічні вісті" – 2008. – №1 – с.46-49.

17. *Singh J. Comparative Analysis of Existing Latest Microcontroller Development Boards / J. Singh, V. Singh // Emerging Research in Electronics, Computer Science and Technology / J. Singh, V. Singh. – Singapore: Springer International Publishing Switzerland, 2015. – P. 1011–1026.*

18. Бойченко С. В., Іванченко О. В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – Київ: НАУ, 2017. – 63 с.

19. Круглік Є. І., Харьков М. В. Актуальні проблеми віртуалізації в *DOCSIS* мережах : Матеріали III Міжнародної науково-практичної конференції «Сучасні тенденції розвитку науки», м. Київ, 9-10 березня 2019 р. Київ, 2019. с. 36-38.

20. Харьков М. В. Використання кластерних технологій для підвищення відмовостійкості комп'ютерних систем : матеріали Міжнародної наукової інтернет-конференції (випуск 38), м. Тернопіль, 7 травня 2019 р. Тернопіль, 2019. с. 100-101.

21. ГОСТ 2.301-68. Единая система конструкторской документации. Форматы. – Введ. 2002–01–01. – М. : Вид.-во стандартов, 2006. – 27 с.

22. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86 с.

Додаток А

Лістинг програми для ПК

```
public partial class MainForm : Form {
EngineController engine; LoadController load; FileController fileRW;
FanController leftFan; FanController rightFan; int chartSecondsCounter = 0;
List<Point2D> chartDataTemp1 = new List<Point2D>();
List<Point2D> chartDataTemp2 = new List<Point2D>();
List<Point2D> chartDataLoad = new List<Point2D>();
int executeTimeCounter = 0; long keepAliveTimeCounter = 0; Thread dataCollector;
bool isPlanInExecution = false, isChartBeingRendered = false;
bool isMCFound = false, stopCollectingData = false; string threadStrDbg = "";
// Commands to MC
readonly byte spResetConnection = 0x31, spStart = 0x32;
readonly byte spAllOk = 0x33, spStartEngine = 0x34;
readonly byte spEngineRPM = 0x35, spStopEngine = 0x36;
readonly string initializedStr = "INITIALIZED", notInitializedStr = "MC";
long intermediateTemp1 = 0, intermediateTemp2 = 0, intermediateRPM = 0;
void buttonControlEngineStop_Click(object sender, EventArgs e) {
    if(!serialPortMC.IsOpen) {
    isMCFound = false; keepAliveTimeCounter = 0; timerChartInfo.Stop();return;}
    engine.On = false; textBoxCurrentEngineOn.Text = "Hi";
    serialPortMC.BaseStream.WriteByte(spStopEngine); }
private void buttonControlEngineSetValues_Click(object sender, EventArgs e) {
    if (!serialPortMC.IsOpen){isMCFound=false; keepAliveTimeCounter = 0;
        timerChartInfo.Stop(); return; }
    engine.RPM = (int)numericUpDownControlEngineRPM.Value;
    textBoxControlEngineCurrentRPM.Text = engine.RPM.ToString();
    textBoxControlEngineCurrentVoltage.Text = ((int)engine.Voltage).ToString();
    serialPortMC.BaseStream.WriteByte(spEngineRPM);
```

```
        byte[] buffer = BitConverter.GetBytes(engine.RPM);
        serialPortMC.BaseStream.Write(buffer, 0, buffer.Length); }
private void buttonControlEngineStart_Click(object sender, EventArgs e) {
        if (!serialPortMC.IsOpen){isMCFound =false;keepAliveTimeCounter=0;
timerChartInfo.Stop();return;}engine.On=true;textBoxCurrentEngineOn.Text="Так";
serialPortMC.BaseStream.WriteByte(spStartEngine);}
private void buttonLoadFileFromPath_Click(object sender, EventArgs e) {
        if (openFileDialogPlan.ShowDialog() == DialogResult.OK) {
        if (fileRW.LoadFromFile(openFileDialogPlan.FileName)) {
                listViewPlanRecords.Items.Clear(); int idx = 0;
                foreach (PlanRecord pl in fileRW.GetRecords()) {
listViewPlanRecords.Items.Add(new
ListViewItem(idx.ToString(),pl.GetTimeStr(),pl.GetSpeed().ToString(),pl.GetLoad().ToStri
ng(),pl.GetT1().ToString(),pl.GetT2().ToString()));idx++;} }
private void buttonSavePlan_Click(object sender, EventArgs e) {
        if (saveFileDialogPlan.ShowDialog() == DialogResult.OK) {
        List<string> list = new List<string>(listViewPlanRecords.Items.Count);
        foreach (ListViewItem item in listViewPlanRecords.Items) {
list.Add(FileController.ApplyTemplate(item.SubItems[1].Text,item.SubItems[2].Text,item.
SubItems[3].Text,item.SubItems[4].Text,item.SubItems[5].Text));}
        if (!fileRW.SaveToFile(saveFileDialogPlan.FileName, list))
                MessageBox.Show("Save error", "", MessageBoxButtons.OK,
MessageBoxIcon.Warning); }}
private async void buttonStartSelected_Click(object sender, EventArgs e) {
        if (listViewPlanRecords.SelectedItems.Count > 0) {
Dictionary<int,PlanRecord>recordsToExecute=Convert_ListViewItems_ToList();
//Deleting records not in execution list
int executeFromIdx = int.Parse(listViewPlanRecords.SelectedItems[0].SubItems[0].Text);
foreach (var toDelete in recordsToExecute.Where(kv => kv.Key <
executeFromIdx).ToList())
```

```
recordsToExecute.Remove(toDelete.Key); await ExecutePlanAsync(recordsToExecute);
} else MessageBox.Show("Record not chosen", "", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);}
private async Task ExecutePlanAsync(Dictionary<int, PlanRecord> recordsToExecute) {
SetAllControlToInitial();PlanInExecutionControlDisabler(true);executeTimeCounter=0;
isPlanInExecution = true; Dispatcher uiDispatcher = Dispatcher.CurrentDispatcher;
await Task.Run(() => //task is synchronous for itself {
DateTime recordEndTime = DateTime.Now; DateTime recordStartTime;
foreach (KeyValuePair<int, PlanRecord> recordKV in recordsToExecute) {
recordStartTime = recordEndTime;
recordEndTime = recordEndTime.Add(recordKV.Value.GetTime());
executeTimeCounter = (int)recordEndTime.Subtract(recordStartTime).TotalSeconds;
var uiDispatcherTask = uiDispatcher.BeginInvoke(new Action(() => {
textBoxRecordInExecution.Text = recordKV.Key.ToString();
textBoxStartTime.Text = recordStartTime.ToString("hh:mm:ss tt");
textBoxEndTime.Text = recordEndTime.ToString("hh:mm:ss tt");
textBoxRemainingTime.Text = executeTimeCounter.ToString();
timerCurrentRecord.Start();}), null); uiDispatcherTask.Wait();
while (DateTime.Now < recordEndTime) {if (!isPlanInExecution) return;
Thread.Sleep(1000); }
} }); PlanInExecutionControlDisabler(false); SetAllControlToInitial();
MessageBox.Show("Stop plan execution", "", MessageBoxButtons.OK,
MessageBoxIcon.Information); }
private void buttonStopPlan_Click(object sender, EventArgs e) {
isPlanInExecution = false; }
private void timerCurrentRecord_Tick(object sender, EventArgs e) {
if (executeTimeCounter > 0 && isPlanInExecution) {
executeTimeCounter--; textBoxRemainingTime.Text = executeTimeCounter.ToString();
} else  timerCurrentRecord.Stop(); }
private void timerChartInfo_Tick(object sender, EventArgs e) {
```

```
if (isChartBeingRendered) {
chartSecondsCounter++;
//Collect info
chartTemperature.Series["T1"].Points.AddXY(chartSecondsCounter, Interlocked.Read(ref
intermediateTemp1));
chartTemperature.Series["T2"].Points.AddXY(chartSecondsCounter, intermediateTemp2);
//Write to tempdata
chartDataTemp1.Add(new
Point2D((int)chartTemperature.Series["T1"].Points.Last().XValue,
(int)chartTemperature.Series["T1"].Points.Last().YValues[0]));
chartDataTemp2.Add(new
Point2D((int)chartTemperature.Series["T2"].Points.Last().XValue,
(int)chartTemperature.Series["T2"].Points.Last().YValues[0]));
int newLoadValue = -100; if (isPlanInExecution) {
int recordIdx = int.Parse(textBoxRecordInExecution.Text);
newLoadValue = int.Parse(listViewPlanRecords.Items[recordIdx].SubItems[3].Text);
} else  newLoadValue = load.Load;
chartLoad.Series["Load"].Points.AddXY(chartSecondsCounter, newLoadValue);
chartDataLoad.Add(new Point2D((int)chartLoad.Series["Load"].Points.Last().XValue,
(int)chartLoad.Series["Load"].Points.Last().YValues[0]));}
textBoxT1.Text = intermediateTemp1.ToString(); textBoxT2.Text =
intermediateTemp2.ToString();
textBoxSpeed.Text = intermediateRPM.ToString(); if(intermediateRPM > 20000) {
MessageBox.Show($"Resolution number limit exceeded, engine will be stopped to prevent
it break down", "WARNING", MessageBoxButtons.OK, MessageBoxIcon.Warning);
buttonControlEngineStop_Click(sender, e); }}
private async void buttonSaveCharts_Click(object sender, EventArgs e) {
if (isChartBeingRendered) {
MessageBox.Show("To write data from chart to file drawing should be stopped ", "",
MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
return;}
string dirName = DateTime.Now.ToString("dd.MM.yy hh-mm tt"); // dd/mm/yy hh:mm:ss
tt
if (Directory.Exists(dirName)) {
DialogResult dRes = MessageBox.Show($"Directory {dirName} already exist\nRewrite?",
"", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
if(dRes == DialogResult.No)  return; Directory.Delete(dirName, true);
await Task.Run(() => {
DirectoryInfo dirInfo = Directory.CreateDirectory(dirName);
FileStream temp1FileStream = File.Create(dirInfo.FullName + @"\Temp11.txt");
FileStream temp2FileStream = File.Create(dirInfo.FullName + @"\Temp22.txt");
FileStream loadFileStream = File.Create(dirInfo.FullName + @"\Load.txt");
// writing temp1
for (int i = 0; i < chartSecondsCounter; i++) {
WriteTextToFile(temp1FileStream, chartDataTemp1[i].ToString() + "\r\n");
WriteTextToFile(temp2FileStream, chartDataTemp2[i].ToString() + "\r\n");
WriteTextToFile(loadFileStream, chartDataLoad[i].ToString() + "\r\n");}
// Release files
temp1FileStream.Close(); temp2FileStream.Close(); loadFileStream.Close();});
MessageBox.Show($"Data written into directory {dirName}", "", MessageBoxButtons.OK,
MessageBoxIcon.Information);}
private static void WriteTextToFile(FileStream fs, string text) {
byte[] info = new UTF8Encoding(true).GetBytes(text); fs.Write(info, 0, info.Length);}
private async Task ScanForMC() { await Task.Run(async () =>  {
string[] ports = SerialPort.GetPortNames(); foreach (string p in ports) {
MessageBox.Show($"Scanning port {p}"); // DEBUG REMOVE THIS
serialPortMC = new SerialPort(p, 9600); if (MCDetected()) {
MessageBox.Show($"MC not found on port {p}");
isMCFound = true; timerChartInfo.Start(); break;}
System.Threading.Thread.Sleep(1000); // wait a lot after closing serial}
```

```
if (!isMCFound) {
MessageBox.Show($"MC not found\nFor program to work correctly connect MC and push
'Connect to LCP1768 button'", "Attention", MessageBoxButtons.OK,
MessageBoxIcon.Warning);return;}
serialPortMC.BaudRate = 9600; serialPortMC.DtrEnable =
true;serialPortMC.ReadTimeout = 1000; try {
serialPortMC.Open();
System.Threading.Thread.Sleep(1000); // Need to wait before port is open....
serialPortMC.DiscardInBuffer();
//await serialPortMC.BaseStream.WriteAsync(buffer, 0, buffer.Length); // Using
writeAsync of baseStream, as SerialPort lib is broken...
serialPortMC.BaseStream.WriteByte(spStart); System.Threading.Thread.Sleep(500);
int retryInitializeCounter = 0; // Max is 10 int bigRetryCounter = 0; // Max is 3
while (true) { if (retryInitializeCounter >= 10) { bigRetryCounter++;
if(bigRetryCounter >= 3) {
MessageBox.Show("Cannot initialize data fetch from sensors\nReload LCP1768 and
check it’s state", "Criticali error", MessageBoxButtons.OK, MessageBoxIcon.Error);
isMCFound = false; /*UI should be turned off*/ timerChartInfo.Stop(); break;}
retryInitializeCounter = 0; serialPortMC.BaseStream.WriteByte(spResetConnection);
System.Threading.Thread.Sleep(1000); serialPortMC.DiscardInBuffer();
serialPortMC.BaseStream.WriteByte(spStart); System.Threading.Thread.Sleep(500);}
string returnMessage = serialPortMC.ReadLine(); if
(returnMessage.Contains(initializedStr)) break;
else retryInitializeCounter++;} // Successful handshake, collecting data
dataCollector = new Thread(new ThreadStart(CollectSensorsData));
dataCollector.Start(); } catch (Exception ex) {
MessageBox.Show($"Error occured. Ensure that LCP1768 is connected and push Connect
to LCP1768, if all seems ok\nError message:\n{ex.Message}",
"Reading sensors data error",MessageBoxButtons.OK, MessageBoxIcon.Error);
isMCFound = false; if(serialPortMC.IsOpen) {
```

```
serialPortMC.DiscardInBuffer(); serialPortMC.Close(); }
timerChartInfo.Stop(); } }); } private bool MCDetected() { try {
serialPortMC.Open(); System.Threading.Thread.Sleep(1000); // just wait a lot
string returnMessage = serialPortMC.ReadExisting();
// in MC sketch should be Serial.println("MC") inside  void loop()
if (returnMessage.Contains(notInitializedStr))  return true; else  false; }
catch (Exception ex) { MessageBox.Show($"Exception {ex}"); return false;
} finally { serialPortMC.Close(); } }
private async void buttonScanPorts_Click(object sender, EventArgs e) {
EnableAllUI(false); await ScanForMC();
if(isMCFound) EnableAllUI(true); } private void CollectSensorsData() {
System.Timers.Timer aTimer = new System.Timers.Timer(); aTimer.Interval = 1000;
aTimer.Elapsed += new ElapsedEventHandler(OnTimedEvent); aTimer.Start();
serialPortMC.DiscardInBuffer(); // Discarding because data in buffer is obsolete
while (serialPortMC.IsOpen && !stopCollectingData) { try {
threadStrDbg += DateTime.Now.ToString("mm:ss:ff") + '\n';  //REMOVE
if (returnMessage.Contains(notInitializedStr)) {
MessageBox.Show("Communication protocol was corrupted\Reinitialize connection by
pushing 'Connect to MC'", "Error ", MessageBoxButtons.OK, MessageBoxIcon.Error);
break;}string[]allData=returnMessage.Split(' '); foreach(string kv in allData) {
string[] keyValue = kv.Split(':');
if(keyValue[0] == "Temp1") Interlocked.Exchange(ref intermediateTemp1,
(long)float.Parse(keyValue[1]));
if (keyValue[0] == "RPM") Interlocked.Exchange(ref intermediateRPM,
(long)float.Parse(keyValue[1])); }
if (Interlocked.Read(ref keepAliveTimeCounter) >= 20) {
System.Threading.Thread.Sleep(1000); serialPortMC.BaseStream.WriteByte(spAllOk);
System.Threading.Thread.Sleep(1000);keepAliveTimeCounter = 0; }}catch(Exception e){
MessageBox.Show(e.ToString());
```

MessageBox.Show("Communication protocol was corrupted\Reinitialize connection by pushing 'Connect to MC'", "Error ", MessageBoxButtons.OK, MessageBoxIcon.Error); serialPortMC.Close(); }}

aTimer.Stop(); isMCFound = false; keepAliveTimeCounter = 0; timerChartInfo.Stop();}

private void MainForm_FormClosed(object sender, FormClosedEventArgs e)

{stopCollectingData = true;}

private void OnTimedEvent(object source, ElapsedEventArgs e)

{Interlocked.Increment(ref keepAliveTimeCounter);}}

Додаток Б

Лістинг програми для мікроконтролера

```
#include <OneWire.h>
OneWire  ds(10);
int Relay1 = 5;
int Semistor = 3;
#define SECOND 1000 //ms
#define LOOP_ITERATION_TIME 910
#define TIMEOUT_ITERATIONS 45
unsigned int TIMEOUT_MAX = ((unsigned
int)(TIMEOUT_ITERATIONS*SECOND)/LOOP_ITERATION_TIME);
#define NOT_FOUND_MAX 5
char notInitializedMsg[8] = "MC";
char initializedMsg[12] = "INITIALIZED";
// Commands From Serial
#define RESET_CONNECTION 0x31
#define START 0x32
#define ALL_OK 0x33
#define START_ENGINE 0x34
#define ENGINE_RPM 0x35
#define STOP_ENGINE 0x36
void setup(void) {
  Serial.begin(9600); pinMode(Relay1, OUTPUT);
  pinMode(Semistor, OUTPUT); }
void loop(void) {
    if(!isInitialized) {
      Serial.println(notInitializedMsg); delay(750);
    } else {
```

```
    if(notFoundCounter >= NOT_FOUND_MAX) {
      ResetControlInfo();
      // MC will start printing notInitializedMsg and SMC-2 will try to re-connect
      return; }
    if(timeoutCounter >= TIMEOUT_MAX) { ResetControlInfo(); return; }
    timeoutCounter++;
    PrintTempSensorsDataToSerial();
    // ENGINE CONTROL
    val = analsemistorLimitationead(A0);
    RPMsensorValue = map(val, 0, 1023, minRPM, maxRPM); // mapping sensor to
min and max RPM
    tolerance = (analsemistorLimitationead(A1)+1);
    rew = RPMsensorValue - tolerance;
    if (rew < 0) rew = 0;
    if (val > 0) {
      if ( holl >= 1) {
        RPMsensorValueOb = 60000000 / ((tic * 0.0625 ) * kImp / 10);
        if ( RPMsensorValueOb >= 0) realRotations = RPMsensorValueOb ;
        semistorLimitation = map(val, 0, 1023, semistorLimitationForMinRotation, 7);
// for how much time semistor could be open
        dimming = map(realRotations, rew , (RPMsensorValue + tolerance),
semistorLimitation, minSemistorStartRotation); //calculate semistor control
        holl = 0;
      }
      if (tic == 0)  dimming = semistorValueOfInitialImpulce ;
      dimming = constrain(dimming, semistorLimitation, minSemistorStartRotation) ;
    } else  dimming = 130;
    int dimtime = (75 * dimming);
    tims = micros();
    if (tims >= (time + dimtime)) {
```

```
      digitalWrite(Semistor, HIGH);          // open semistor
      delayMicroseconds(10);                 // 10ms delay
      digitalWrite(Semistor, LOW);             // stop signalling semistor
    }
    currentTime = millis();
    if (currentTime >= (loopTime + 1000)) {
      Serial.print("RPM:");
      Serial.println(RPMsensorValue);
      sp = 0;
      loopTime = currentTime; }} ReadCommandFromSerial(); }
void ReadCommandFromSerial() {
  char incomingCommand;
  if(Serial.available() > 0) {
    incomingCommand = Serial.read();
    switch(incomingCommand) {
      case RESET_CONNECTION: ResetControlInfo(); Serial.println("Reseting
Config"); break;
      case START: isInitialized = true; Serial.println(initializedMsg); break;
      case ALL_OK: timeoutCounter = 0; break;
      case START_ENGINE: digitalWrite(Relay1, LOW); break;
      case ENGINE_RPM: int newRpm = Serial.read(); break;
      case STOP_ENGINE: digitalWrite(Relay1, HIGH); break;
      default: break; }}}
void PrintTempSensorsDataToSerial() {
  byte i; byte present = 0;
  byte type_s; byte data[12]; byte addr[8];
  float celsius, fahrenheit;
  if ( !ds.search(addr)) {
    notFoundCounter++;
    ds.reset_search();
```

```
  delay(250);
  return;}
      if (OneWire::crc8(addr, 7) != addr[7]) {
    notFoundCounter++;
    return; }
// the first ROM byte indicates which chip
switch (addr[0]) {
  case 0x10: type_s = 1; break;
  case 0x28: type_s = 0; break;
  case 0x22: type_s = 0; break;
  default: notFoundCounter++; return; }
ds.reset(); ds.select(addr); ds.write(0x44); delay(850);
present = ds.reset(); ds.select(addr); ds.write(0xBE);// Read Scratchpad
for ( i = 0; i < 9; i++)  // we need 9 bytes
  data[i] = ds.read();
int16_t raw = (data[1] << 8) | data[0];
if (type_s) {
  raw = raw << 3; // 9 bit resolution default
  if (data[7] == 0x10) {
    // "count remain" gives full 12 bit resolution
    raw = (raw & 0xFFF0) + 12 - data[6];
  } } else {
  byte cfg = (data[4] & 0x60);
  // at lower res, the low bits are undefined, so let's zero them
  if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
  else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
  else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
  // default is 12 bit resolution, 750 ms conversion time }
celsius = (float)raw / 16.0;
fahrenheit = celsius * 1.8 + 32.0;
```

```
Serial.print("Temp1:");
Serial.println(celsius);
notFoundCounter=0;
ds.reset_search(); }
```