КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Інститут високих технологій

Завідувач кафедри нанофізики конденсованих середовищ
проф. Валерій Антонович Скришевський
Протокол №____засідання кафедри
від “____”_____2021 р.

АВТОМАТИЗАЦІЯ УСТАНОВКИ ДЛЯ ВИМІРЮВАННЯ СПЕКТРІВ ФОТО- ТАЕЛЕКТРОЛЮМІНЕСЦЕНЦІЇ

Випускна кваліфікаційна робота магістра

студента напрямку підготовки

105 «Прикладна фізика та наноматеріали»

Шапіренка Олександра Юрійовича

Науковий керівник:

кандидат фіз.-мат. наук, доцент

Шкавро Анатолій Григорович

Оцінка захисту роботи

_____

КИЇВ - 2021

Реферат

Магістерська робота: 55 с., 11 рис., 8 блок-схем, 2 графіки, 12 посилань(з них 3 - електронні), 6 доданків.

Метою роботи була модернізація та оптимізація експериментальної установки для вимірювання спектрів фото- та електролюмінесценції.

У ході роботи було замінено старий блок керування кроковими двигунами на платі ET3401 на новий, який працює на мікроконтролері ATMega328P у складі платформи Arduino Uno. Написано код для керування кроковими двигунами, яким був запрограмований відповідний мікроконтролер. Також було покращено параметри імпульсів ФЕПа та розширено його динамічний діапазон, впроваджено автономну систему термостабілізації ФЕПа, оптимізовано та доповнено програми та алгоритми для виконання вимірів. Написано програму для обміну інформації між мікроконтролером та ПК через COM-порт. Виконано апробацію установки.

Ключові слова: **АВТОМАТИЗОВАНА УСТАНОВКА, МІКРОКОНТРОЛЕР, ARDUINO UNO, КРОКОВИЙ ДВИГУН, ФЕП, ФОТОЛЮМІНЕСЦЕНЦІЯ, ТЕРМОСТАБІЛІЗАЦІЯ**

Зміст

Вступ

Явище люмінісценції отримало широке застосування у науці, техніці та сучасному побуті. Також воно являє собою один з найпотужніших засобів дослідження фізичних процесів та характеристик конденсованих середовищ. Це обумовлено тим, що люмінісценція тісно пов'язана з зонною структурою твердого тіла, відображає особливості релаксації електронних збурень зовнішнім випромінюванням, дозволяє встановити енергетичну схему локальних домішкових центрів та дефектів кристалічної структури.

В лабораторії вже є експериментальна установка, яка протягом багатьох років успішно використовувалась для вимірів і дослідження спектрів лімінесценції. При цьому з плином часу деякі складові частини установки застаріли і вже існують доступні технології, які можна набагато зручніше імплементувати та використовувати з тими ж цілями. В установці для автоматизації вимірів використовувалась модифікована плата ET3401, яка підключається до комп'ютера шиною ISA. Сучасні комп'ютери не підтримують дану шину. І взагалі реалізація керування кроковими двигунами з використанням цієї плати виходить досить громіздка. Як аналог, який нічим не буде поступатись старому блоку керування кроковими двигунами моїми колегами було розроблено новий блок керування з мікроконтролером у складі платформи Arduino Uno. Це рішення дозволяє прискорити виконання керування кроковими двигунами, і робить його більш автономним, адже операції будуть виконуватись збоку мікроконтролера, а не з самого комп'ютера.

Для впровадження нового блоку необхідно створити програмний комплекс для обміну інформації між платою та комп'ютером та запрограмувати мікроконтролер на виконання необхідних функцій. Також, так як ми прагнемо перейти до більш сучасних приладів і технологій необхідно замінити комп'ютер

на новий з сучасними портами підключення периферійних пристроїв та апаратним забезпеченням.

Наступним кроком є покращення параметрів вимірювання, таких як динамічний діапазон ФЕПа, який виконує роль оптичного приймача в установці. Це робиться з метою покращення точності вимірювання для вже імплементованих алгоритмів проведення експериментів, що потенційно надасть можливість вимірювати більш швидкотривалі фізичні ефекти такі як кінетика затухання люмінесценції. При цьому треба зазначити, що для досягнення цієї цілі необхідна заміна АЦП, який зараз використовується в установці на більш швидкодіючий(з більшою частотою дискретизації). Всі існуючі алгоритми при цьому міняти не потрібно.

Метою роботи є модернізація установки та покращення параметрів вимірювання.

Огляд літератури

1.1. Будова гібридного крокового двигуна

Гібридні двигуни є більш дорогими, ніж двигуни з постійними магнітами, зате вони забезпечують меншу величину кроку, більший момент і велику швидкість. Типове число кроків на оберт для гібридних двигунів становить від 100 до 400 (кут кроку 3.6 - 0.9 °). Гібридні двигуни поєднують в собі перевагу двигунів зі змінним магнітним опором і двигунів з постійними магнітами. Ротор гібридного двигуна має зубці, розташовані в осьовому напрямку(Рис.1.1.)

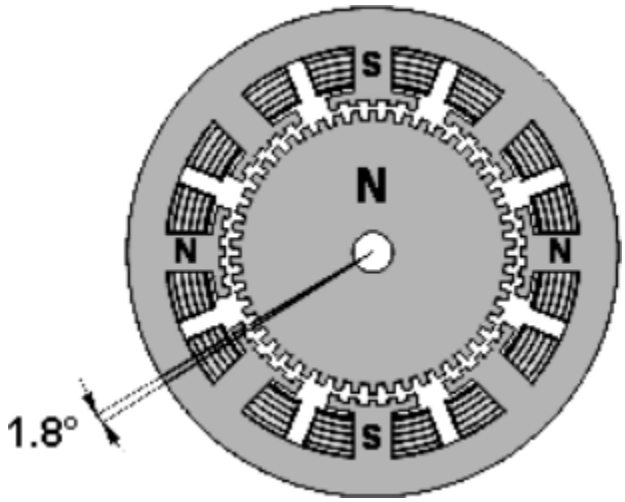


Рис.1.1. Гібридний двигун

Ротор розділеній на дві частини, між котрими розташований циліндричний постійнім магніт. Таким чином, зубці верхньої половинки ротора є північними полюсами, а зубці нижньої половинки - південними. Крім того, верхня і нижня половинки ротора повернені один відносно одного на половину кута кроку зубців. Число пар полюсів ротора дорівнює кількості зубців на одній з його половинок. Зубчасті полюсні наконечники ротора, як і статор, набрані з окремих пластин для зменшення втрат на вихрові струми.

Статор гібридного двигуна також має зубці, забезпечуючи велику кількість еквівалентних полюсів, на відміну від основних полюсів, на яких розташовані обмотки. Зазвичай використовуються 4 основних полюса для 3,6°. двигунів і 8 основних полюсів для 1.8°- 0.9°. двигунів. Зубці ротора забезпечують менший опір магнітного кола в певних положеннях ротора, що покращує статичний і динамічний момент. Це забезпечується відповідним розташуванням зубців, коли частина зубців ротора знаходиться строго напроти зубців статора, а частина між ними. Ротор зображений на малюнку 4 має 100 полюсів (50 пар), двигун має 2 фази, тому загальна кількість полюсів - 200, а крок, відповідно, 1,8°.

Поздовжній переріз гібридного крокового двигуна показано на рис.1.2. Стрілками показано напрямок магнітного потоку постійного магніту ротора. Частина потоку (на малюнку показана чорною лінією) проходить через полюсні накінечники ротора, повітряні зазори і полярний накінечник статора. Ця частина не бере участі в створенні моменту.

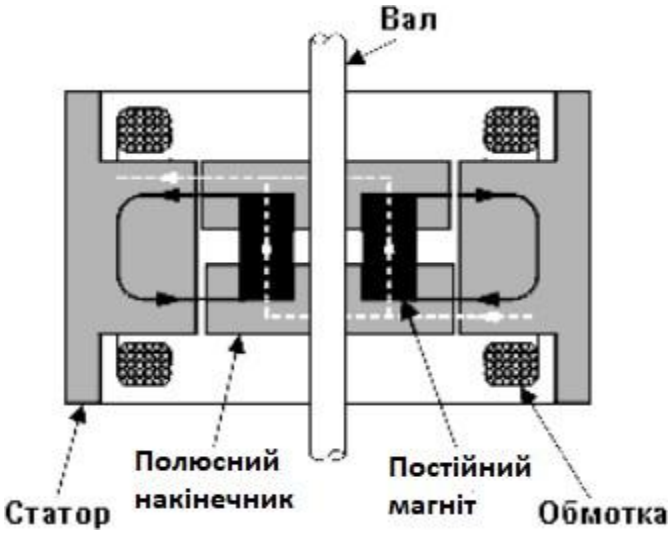


Рис.1.2. Поздовжній переріз гібридного крокового двигуна

Як видно з рис. 5, повітряні зазори у верхнього і нижнього полюсного накінечника ротора різні. Це досягається завдяки повороту полюсів накінечників на половину кроку зубців. Тому існує інше магнітне коло, яке містить мінімальні повітряні зазори і, як наслідок, має мінімальний магнітний опір. З цього кола замикається інша частина потоку (на рис. 1.2. показана штрихованою білою лінією), яка і створює момент. Частина кола лежить в площині, перпендикулярній малюнку, тому не зображена на малюнку[1].

1.2. Способи керування кроковим двигуном. Напівкроковий режим

Існують чотири основні режими керування кроковими двигунами:

- Повнокроковий режим без перекриття фаз
- Повнокроковий режим з перекриттям фаз
- Напівкроковий режим
- Мікрокроковий режим

Перший спосіб забезпечується за допомогою позмінної комутації фаз, при цьому вони не перекриваються. В один момент часу включена тільки одна фаза (рис. 1.3а). Точки рівноваги ротора для кожного кроку збігаються з "природними" точками рівноваги ротора у не включеного двигуна. Недоліком цього способу керування є те, що для біполярного двигуна в один і той же момент часу використовується 50% обмоток, а для уніполярного - тільки 25%. Це означає, що в такому режимі не може бути отриманий повний момент.

Другий спосіб - управління фазами з перекриттям: дві фази включені в один і той же час. Ротор фіксується в проміжних позиціях між полюсами статора(Рис. 1.3б) і забезпечується приблизно на 40% більший момент, ніж в разі однією включеною фази. Цей спосіб управління забезпечує такий же кут

кроку, як і перший спосіб, але позиція точок рівноваги ротора зміщена на пів-кроку.

Напівкроковий режим - комбінація двох попередніх, коли двигун робить крок в половину основного (рис. 1.3в). Кожен другий крок включена лише одна фаза, а в решті випадків включені дві. В результаті кутове переміщення ротора становить половину кута кроку для перших двох способів управління. Крім зменшення розміру кроку цей режим дозволяє частково позбутися явища резонансу.

У порівнянні з першими двома режимами, напівкроковий має такі переваги:

• більш висока роздільна здатність без застосування більш дорогих

двигунів;

• менші проблеми з явищем резонансу. Резонанс призводить лише до часткової втрати моменту, що зазвичай не заважає нормальній роботі двигуна.

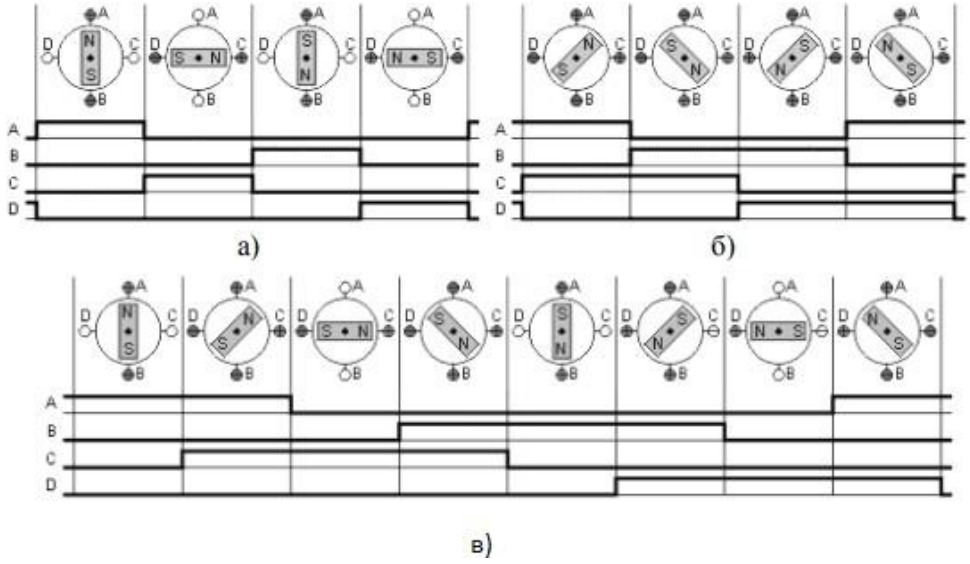


Рис. 1.3. Послідовність комутації фаз а – хвильова, б – крокова, в - напівкрокова

1.3. Будова, принцип дії ФЕП

Одним з основних приймачів оптичного випромінювання, що застосовуються у приладах фотоспектроскопії та фотометрії є фотоелектронний помножувач (ФЕП) та ПЗЗ лінійка. У сучасному світі в якості фотоприймачів частіше використовуються ПЗЗ лінійки, які мають більш високу квантову ефективність та мають координатну чутливість.

Однак незаперечною перевагою ФЕП є великий динамічний діапазон, висока часова роздільна здатність та краща, ніж у ПЗЗ чутливість у короткохвильовій області.

ФЕП — це прилад, у якому падаюче на нього електромагнітне випромінювання (оптичного діапазону) перетвориться в багаторазово підсилені імпульси струму[2]. Він працює на основі зовнішнього фотоефекту. На відміну від звичайного вакуумного фотодіода імпульс струму, який виникає внаслідок фотоемісії електрона підсилюється у ФЕП в $10^5 \div 10^8$ разів, що цілком достатньо для його реєстрації. Квантова ефективність (відношення числа народжених фотоелектронів до числа квантів, що впали на фотоприймач) для ФЕП становить не більше 10%.

Будову, принцип дії та схему включення ФЕП розглянемо на прикладі одного з найбільш широко застосовних приладів — ФЕП-79 (рис. 1.4).

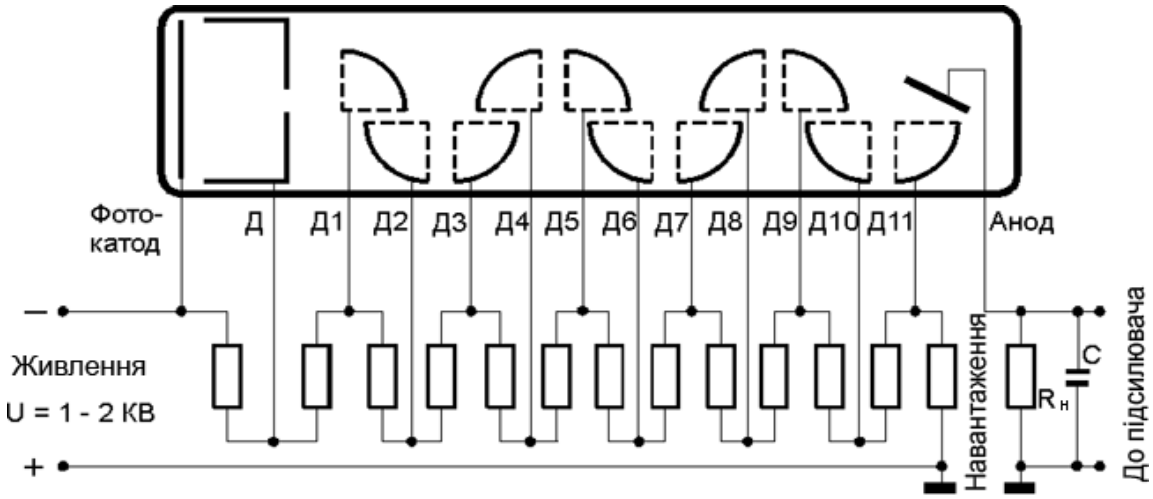


Рис. 1.4. Конструкція та типова схема включення ФЕП-79

ФЕП складається з фотокатоду, катодної камери (Д), динодної системи (Д1 – Д11) та аноду, розміщених в середині вакуумного об'єму [3]. Живлення ФЕП здійснюється від стабілізованого джерела високої напруги $U = -(1 \div 2)$ кВ через резисторний подільник $R1 \div R13$. Зазвичай резистори подільника вибираються однаковими, а його загальний опір становить 2÷10 МОм.

Світловий потік Ф поглинається фотокатодом, який за рахунок фотоефекту емітує у вакуум електрони. В електростатичному полі, створеному електродами катодної камери, фотоелектрони прискорюються й фокусуються на перший динод (Д1). Прискоренний фотоелектрон здатний вибити з поверхні декілька (до 10) вторинних, повільних електронів. Вибиті з першого динода вторинні електрони прискорюються й фокусуються на другий динод. Далі цей процес повторюється на всіх каскадах і з останнього динода підсилений електронний потік збирається анодом.

На аноді електронний потік створює імпульс струму, який реєструється шляхом вимірювання напруги на резисторі навантаження R_H.

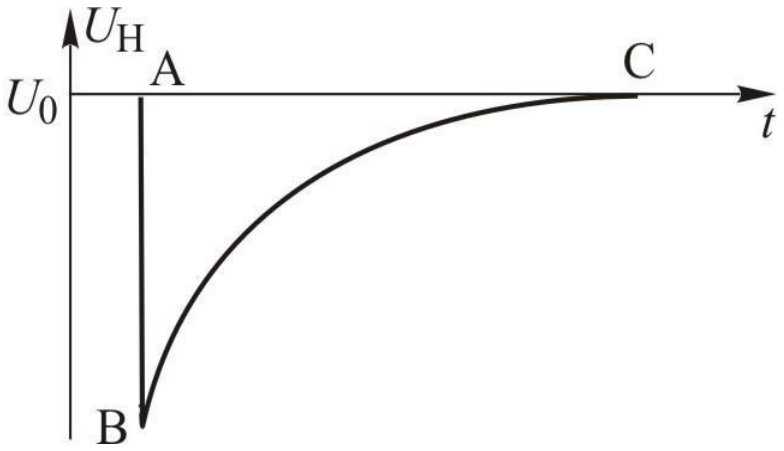


Рис.1.5. Типова форма імпульсу напруги, що падає на резисторі навантаження R_H.

На рис. 1.5 представлено типову форму імпульсу напруги, що падає на резисторі R_H. Тривалість імпульсу напруги залежить від сталої часу *RC*-кола, сформованого резистором R_H та ємністю *C*. Ємність *C* може бути як вхідною ємністю реєструючого приладу, так і спеціально під'єднаною. Мінімально можливе значення тривалості імпульсу становить не більше 10 наносекунд і визначається тими процесами, які відбуваються в самому ФЕП: велика просторова густина електронів спричинює їхнє розштовхування, має місце розкид часу прольоту електронів, які проходять динодну систему по різним траєкторіям, та інше[4].

1.4. Способи реєстрації електричного сигналу с ФЕП

Існує два способи реєстрації електричного сигналу з ФЕП: метод підрахунку фотонів і метод постійний струму. Вибір методу залежить від інтенсивності світлового випромінювання.

У режимі постійного струму вимірювальна система, яка характеризується великим значенням сталої часу, інтегрує всі наявні електричні імпульси на

виході з ФЕП[5]. Вимірюється середнє значення кількості електрики, яка перенесена з динодів на анод за одиницю часу. Обмежень зверху на частоту надходження фотонів тут не виникає. В області слабких світлових потоків усереднення струму стає не ефективним, бо зменшується кількість фотонів за одиницю часу, що потрапляють до ФЕП.

Загальна блок-схема реалізації методу постійного струму представлено на рис. 1.6.

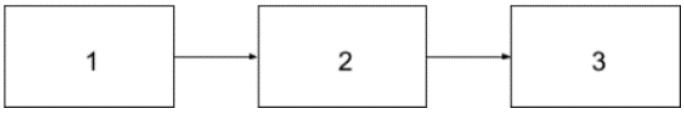


Рис. 1.6. Загальна блок-схема реалізації методу постійного струму.

Електричний сигнал з ФЕП (1) подається на підсилювач постійного струму (2). Підсилювач постійного струму інтегрує всі наявні на вході імпульси за одиницю часу і видає результат на пристрій індикації (3).

За умови слабких світлових сигналів для їх реєстрації застосовується метод підрахунку фотонів[6]. У цьому методі вимірюється не середнє значення струму, що тече з аноду, а кількість електричних імпульсів, спричинених фотоелектронами. Оскільки швидкість емісії фотоелектронів пропорційна інтенсивності світлового потоку, що падає на катод, а кожен фотоелектрон спричиняє імпульс струму в опорі навантаження ФЕП кількість цих імпульсів також буде пропорційною інтенсивності світлового потоку, що падає на катод.

Загальна блок-схема реалізації методу підрахунку фотонів представлено на рис. 1.7.

Рис. 1.7. Загальна блок-схема реалізації методу підрахунку фотонів.

Сигнал (імпульси напруги резистора навантаження аноду) з ФЕП (1), попередньо пройшовши підсилювач (2), потрапляє на дискримінатор (3), основна функція якого — виявлення імпульсів заданих параметрів. Лічильник (4), керований вихідним сигналом дискримінатора, підраховує кількість наявних імпульсів. З виходу лічильника сигнал потрапляє на пристрій індикації (5).

За умови великої інтенсивності світлового потоку зростає ймовірність того, що двом фотонам на вході ФЕП відповідає один імпульс на виході. Це призводить до порушення лінійності залежності кількості імпульсів струму від кількості фотонів, що потрапили на катод ФЕП, а отже необхідності робити відповідні поправки. Однак заряд, що переноситься в імпульсі, який є результатом „злиття" двох імпульсів буде приблизно вдвічі більше, ніж у випадку звичайного імпульсу, тому в області високої інтенсивності світлового потоку більш коректно використовувати метод середнього струму. Для реалізації методу середнього струму опір навантаження R_H обирають достатньо великим, що дозволяє: по-перше дістати достатньо велику амплітуду напруги при протіканні малих імпульсів струму і по-друге збільшити сталу часу $\tau = R_H C$, і тим самим покращити усереднення сигналу.

Для реалізації методу підрахунку фотонів, навпаки, R_H треба вибирати достатньо малим, задля зменшення сталої часу τ, і, відповідно, зменшення ймовірності „злиття" імпульсів. Але в цьому випадку амплітуда імпульсів

напруги на опорі навантаження виявляється недостатньою для їх безпосередньої

реєстрації електронним лічильником. Тому імпульси напруги з опору навантаження R_H необхідно підсилювати за допомогою широкосмугового підсилювача. Але, крім імпульсів струму спричинених фотоелектронами на опорі навантаження спостерігаються „шумові” імпульси малої амплітуди. Заряд, що переноситься цими імпульсами малий і вони не дають суттєвого внеску в середній струм, але їх кількість, за умови малої інтенсивності світлового потоку може виявитись порівняною з кількістю фотоімпульсів. Отже для забезпечення коректного підрахунку фотонів треба використати дискримінацію імпульсів за амплітудою.

1.5. Характеристики Arduino Uno та мікроконтролера ATMega328P

Arduino Uno - це пристрій на основі мікроконтролера ATmega328. В конструктиві пристрою міститься все необхідне для зручної роботи з мікроконтролером: кварцовий генератор на 16 МГц, 14 цифрових входів / виходів 6 з яких можна використовувати як ШІМ-виходи, 6 аналогових входів, , послідовний порт USB, роз'єм для живлення, роз'єм для програмування всередині схеми (ICSP) і кнопка RESET. Для початку роботи з пристроєм достатньо подати живлення від AC/DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою USB-кабелю. Для програмування Arduino використовується інтегроване середовище розробки Arduino IDE, основане на мові C++ з вбудованими функціями для зручної маніпуляції платою. Контролер ATmega328, що використовується у платформі випускається з прошитим завантажувачем*(bootloader)* , що дозволяє завантажувати в мікроконтролер нові програми без необхідності використання зовнішнього програматора. Взаємодія з ним здійснюється за оригінальним протоколу STK500. Розпіновка плафторми Arduino Uno[7] та мікроконтролеру ATmega328 зображені на рис.1.8. та 1.9. відповідно.

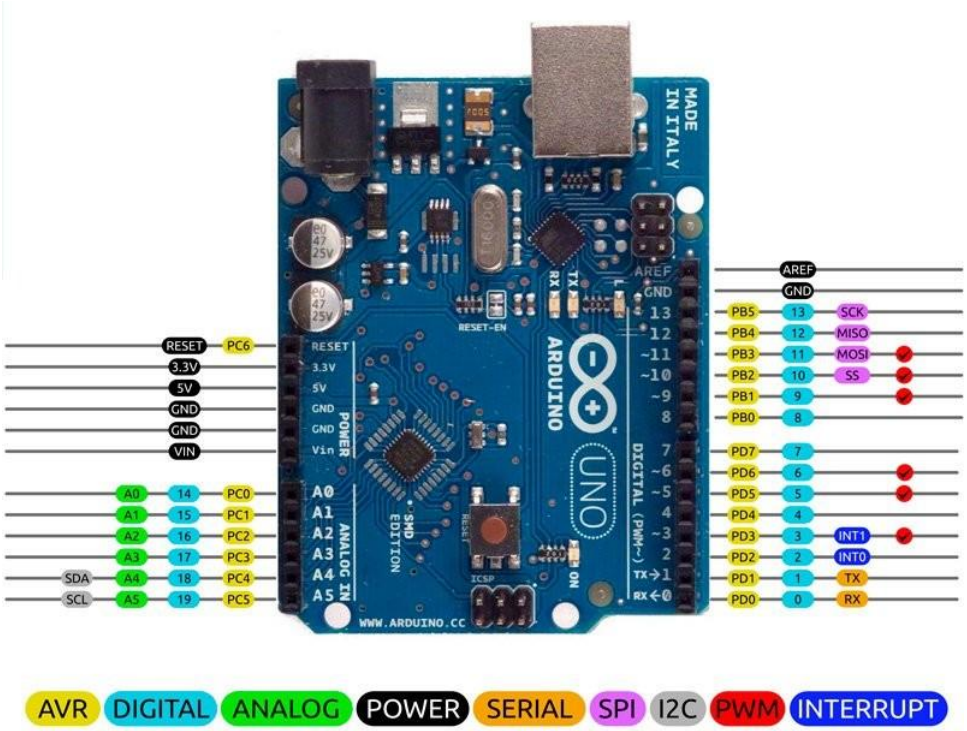


Рис.1.8 Розпіновка Arduino Uno

Характеристики:

- Мікроконтролер: ATmega328P
- Ядро: 8-бітний AVR
- Тактова частота: 16 МГц
- Flash-пам'ять: 32 КБ
- RAM-пам'ять: 2 КБ
- EEPROM-пам'ять: 1 КБ
- Піни введення-виведення: 20
- Піни з перериванням: 2
- Піни з АЦП: 6(Розрядність – 10 біт)
- Піни з ШІМ: 6(Розрядність – 8 біт)
- Апаратні інтерфейси: 1 × UART, 1 × I²C, 1 × SPI
- Напруга логічних рівнів: В
- Вхідна напруга живлення:
    - через USB: 5В
    - через DC-роз'єм або пін Vin: 7,5-12В
- Максимальний вихідний струм Піна 5В: 1 А
- Максимальний вихідний струм Піна 3.3В: 150 мА

- Розміри: 69 × 53 мм

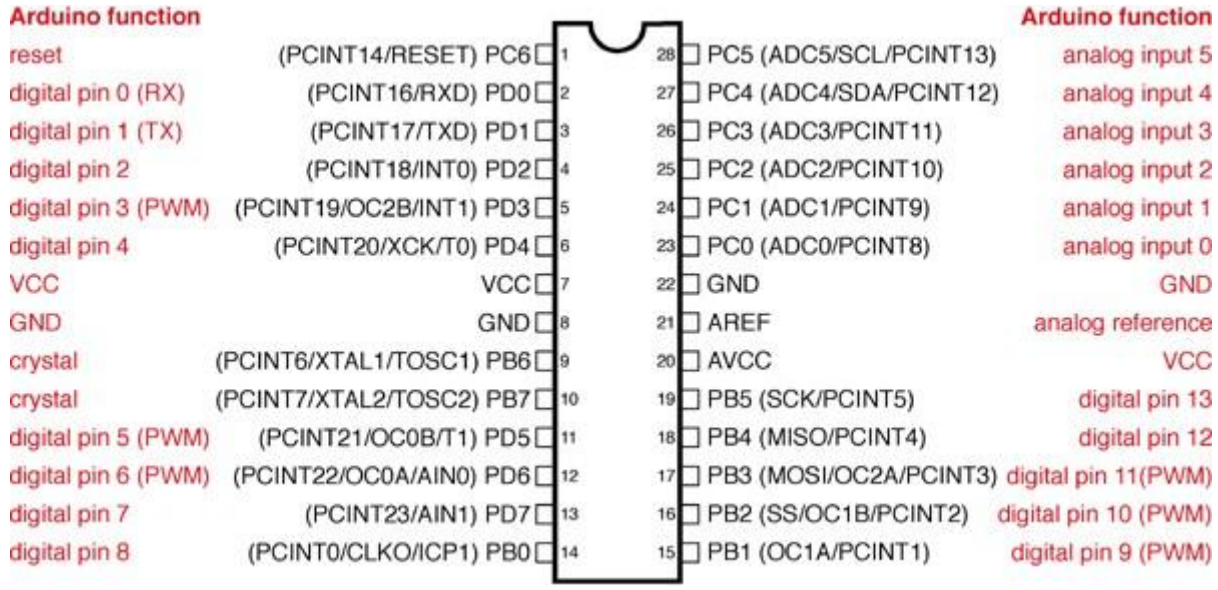


Рис.1.9. Розпіновка ATMega328

ATmega328P - це 8-розрядний мікроконтролер CMOS(комплементарна структура метал-оксид-напівпровідник) із низькою потужністю, заснований на розширеній архітектурі RISC AVR.

Ядро AVR поєднує в собі багатий набір інструкцій з 32 робочими регістрами загального призначення. Всі 32 регістри безпросередньо підключені до арифметичного логічного блоку (ALU), що дозволяє отримати доступ до двох незалежних регістрів в одній інструкції, що виконується за один тактовий цикл.Отримана архітектура є більш ефективною за кодом, досягаючи пропускної здатності до десяти разів більше, ніж звичайні мікроконтролери CISC[8].

Характеристики:

- Пам’ять:
  - 32 kB Flash (програмний простір)
  - 2 kB ОЗУ(оперативна пам’ять)
  - 1 kB EEPROM (постійна пам’ять данних)

- Периферійні функції:
  - Два 8-бітних таймер/лічильника з модулями порівняння та дільниками частоти
  - 16-бітний таймер/лічильник із модулем порівняння та дільником частоти, а також із режимом запису
  - Лічильник реального часу з окремим генератором
  - 6 каналів ШІМ
  - 6-канальний ЦАП із вбудованим датчиком температур
  - Програмиований послідовний порт USART
  - Послідовний інтерфейс SPI
  - Інтерфейс I2C
  - Внутрішня схема порівняння напруги
  - Блок обробки переривань при зміні напруги на виводах мікроконтролера

- Спеціальні функції:
  - Внутрішній генератор тактових імпульсів з можливістю калібрування
  - Обробка внутрішніх та зовнішніх переривань
  - 6 режимів сну (знижене енергоспоживання та зниження шумів для більш точного перетворення АЦП)

- Оцінка швидкості:
  - від 0 до 8 МГц при 2,7 - 5,5 В
  - від 0 до 16 МГц при 4,5 - 5,5 В

- Пакети введення/виведення:
  - 23 програмовані лінії введення/виведення
  - 28-контактний PDIP, 32-вивідному TQFP, 28-накладка QFN / MLF.

1.6. Обмін інформації між пристроями обчислювальної техніки

Одним з найважливіших моментів у роботі апаратних інтерфейсів є синхронізація передачі інформації. Синхронізація - це узгодження процесів взаємодії між пристроями, що полягає в передачі інформації джерелом і її прийому приймачем (одним або декількома). Існують два основних режими синхронізації: синхронний та асинхронний.

У синхронному режимі для синхронізації використовують спеціальну лінію для передачі тактових імпульсів. Інформація в каналі даних зчитується приймачем тільки в ті моменти, коли на лінії синхронізації сигнал активний. Таким чином, зміна станів джерела і приймача взаємонезалежні і виконуються через однакові фіксовані інтервали часу. В цьому випадку приймач повинен встигнути прийняти дані до моменту часу, коли джерело виставить нові дані. Величина фіксованого інтервалу часу синхронізації визначається сумою часів: поширення сигналу в лінії зв'язку, розпізнавання його приймачем і часом фіксації даних[9].Основною перевагою синхронного режиму є відсутність необхідності зворотного зв'язку. Недоліком є відсутність гнучкості, необхідність використання додаткових сигналів.

Найпоширеніший режим, котрий використовується в мікроконтролерах - асинхронний, де один байт даних надсилається як пакет, що містить інформацію про початок і кінець передачі даних, а також інформацію для контролю помилок. Спочатку надсилається не біт даних, а стартовий біт, який вказує на початок передачі даних (запуск пакета даних). Приймач використовує цей біт для процесу синхронного зчитування даних, надісланих стартовим бітом. Після байта даних може йти контрольний біт, що буде використаний для перевірки на коректність отриманих даних.

Після контрольного біту йде стоп-біт, який використовується приймачем для обробки кінця передачі пакета. Асинхронний режим передачі даних показаний на малюнку 1.9. Існує набір параметрів, що повинен бути відомий під час обміну. Одним з них є кількість бітів переданих даних, що визначається типом обладнання(передачі та прийому). Пакет даних на рис. 1.9 містить лише 5 біт даних, при цьому пакети даних зазвичай становлять 8 біт.

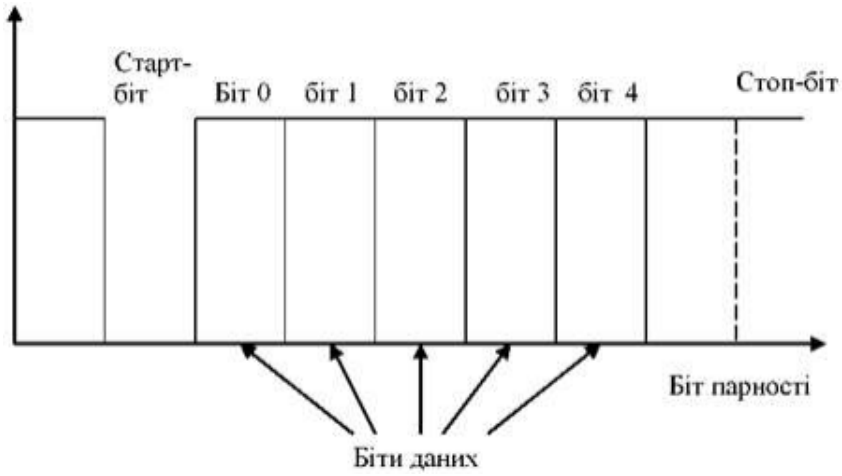


Рис.1.9. Асинхронний режим передачі даних

Майже всі сучасні пристрої використовують для асинхронного обміну даними формат "8-N-1", що означає передачу 8-бітових даних, відсутність біту парності та стоп-біту. Найпопулярніший протокол асинхронного послідовного зв’язку називається "RS-232", і є міжнародним стандартом на сьогоднішній день. Це дуже старий стандарт для підключення комп'ютерів через порт зв'язку (COM). Але навіть зараз стандарт "RS-232" широко використовується у вигляді з'єднання через віртуальний COM-порт при використанні USB[10], саме така технологія і використовується у платах Arduino.

Arduino Uno надає безліч варіантів спілкування з ним через комп’ютер, інший Arduino або взагалі інший мікроконтролер. В ATmega328 є універсальний асинхронний приймач-передавач UART(Universal asynchronous receiver and transmitter), що дозволяє використовувати цифрові виводи: 0 (RX) та 1 (TX) для

послідовної передачі даних та перетворювача USB-UART. Функцію такого перетворювача в платі виконує мікроконтролер ATmega16U2, і при підключені до ПК, дозволяє Arduino відкриватись як віртуальний COM-порт. На платі також існує підтримка послідовних інтерфейсів I2C (TWI) і SPI та відповідних бібліотек для роботи з ними(Wire.h та SPI.h). Для зв'язку з платою Arduino використовується спеціальна програма моніторингу послідовного порту (Serial Monitor), вбудована в програмне забезпечення плати. Монітор послідовної шини відображає дані, що посилаються в плату Arduino через віртуальний послідовний порт за допомогою USB. Для відправки даних вибирається швидкість передачі зі списку, відповідна значенню Serial.begin() у програмі. Для забезпечення зв'язку плати Arduino з комп'ютером або іншими пристроями використовується клас Serial. Цей клас описує змінні, параметри послідовного порту і містить близько 20 функцій для маніпуляції ним.

Практична частина

2.1. Постановка задачі

- Замінити старий блок керування кроковими двигунами, запрограмувати мікроконтролер у новому блоці
- Написати програму для обміну інформацією між комп'ютером та мікроконтролером через COM-порт
- Інвертувати сигнал з ФЕПа та збільшити його динамічний діапазон
- Замінити схему термостабалізації ФЕПа на автономну
- Оптимізувати програми
- Провести апробацію модернізованої установки

2.2. Експериментальна установка. Принцип вимірювання

Блок-схема установки зображена на рис 2.1.

За допомогою лінзи (11) світло джерела випромінювання (10) фокусується на вхідній щілині монохроматора (12). Монохроматор (12) виділяє задану положенням кроковим двигуном(16) довжину хвилі, яка використовується для опромінення зразка (14). З виходу монохроматора (12) світло за допомогою лінзи (13) фокусується на зразку. Для того, щоб на вхід монохроматора (8) потрапляло лише світло фотолюмінесценції, потрібно зразок (14) встановити під таким кутом до оптичної осі монохроматора (12), щоб відбитий від зразка пучок світла, яким здійснюють збудження, не потрапляв на вхід монохроматора (8).

Фотолюмінесцентне світло від зразка (14) збирається на вхідній щілині монохроматора (8) за допомогою лінзи (9). Монохроматор (8) виділяє задану довжину хвилі. Довжина хвилі встановлюється за допомогою крокового

двигуна (7), який підключено до барабану монохроматора (8) і може обертати цей барабан. Роботу крокових двигунів (7,16) контролює блок управління (6), який в свою чергу керується модифікованою платою ET 3401. Цей блок необхідно замінити на новий блок, котрий керується платою Arduino Uno і був створений

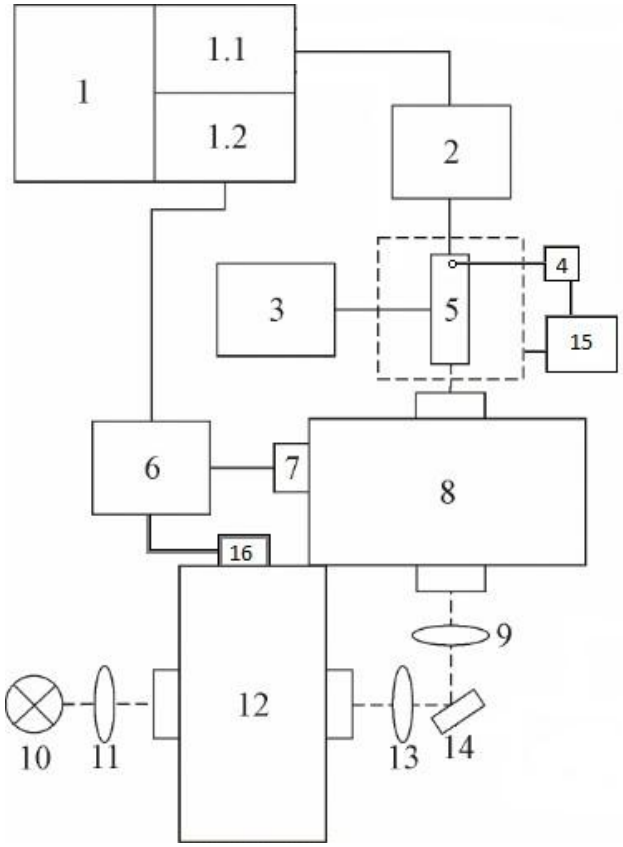


Рис. 2.1. Експериментальна установка вимірювання спектрів фотолюмінесценції:

1 — ЕОМ; 1.1 — плата NQDAC PCI 9112; 1.2 — плата ET 340; 2 — інвертуючий операційний підсилювач; 3 — блок живлення ФЕП; 4 —цифровый терморугулятор W1209; 5 — ФЕП; 6 — блок управління кроковими двигунами; 7,16 — крокові двигуни; 8, 12 — монохроматори; 9,11,

13 — фокусуючи система; 10 — джерело випромінювання; 14 — досліджуваний зразок,15-блок живлення нагріваючого резистора.

моїми колегами.

Вибраний спектральний інтервал фотолюмінесценції з виходу монохроматора (8) потрапляє до фотоелектронного помножувача (5), який перетворює світловий потік в електричний сигнал. ФЕП живиться напругою від блока живлення(3). Охолодження ФЕП здійснюється за допомогою терморегулятора(4), який вмикає або вимикає реле та блоком живлення резистора(15), який занурюється у Посудину Дьюара та випаровує рідкий азот з неї при замиканні реле. Сигнал з ФЕП (5) потрапляє на підсилювач в (2). Підсилений сигнал з ФЕП вимірюється АЦП плати NQDAC PCI 9112 (1.1), після чого результати вимірів обробляється спеціальною програмою ЕОМ (1).

2.3. Заміна блоку керування кроковими двигунами. Програмування мікроконтролера

Новий блок керування на основі платформи Arduino Uno з мікроконтролером ATmega328P був спроектований та зібраний моїми колегами. Компонентами блоку є блок живлення на 9V 2A, понижуючий перетворювач XL 4015, що зменшує напругу від блоку живлення до 4V, спаяна схема ключів для підведення до обмоток двигунів струму необхідного рівня та реалізації напівкрокового режиму керування. Схема ключів кріпиться до плати Arduino зверху. Сигнал з виводів Arduino передається на плату ключів через оптопари. Все це розміщено у одному корпусі з роз’ємами до яких підключаються крокові двигуни та відповідні їм кінцевики.

Вигляд блоку керування зображено на рис.2.2. (Відповідні схеми плати ключів та блоку керування у Доданку 1)

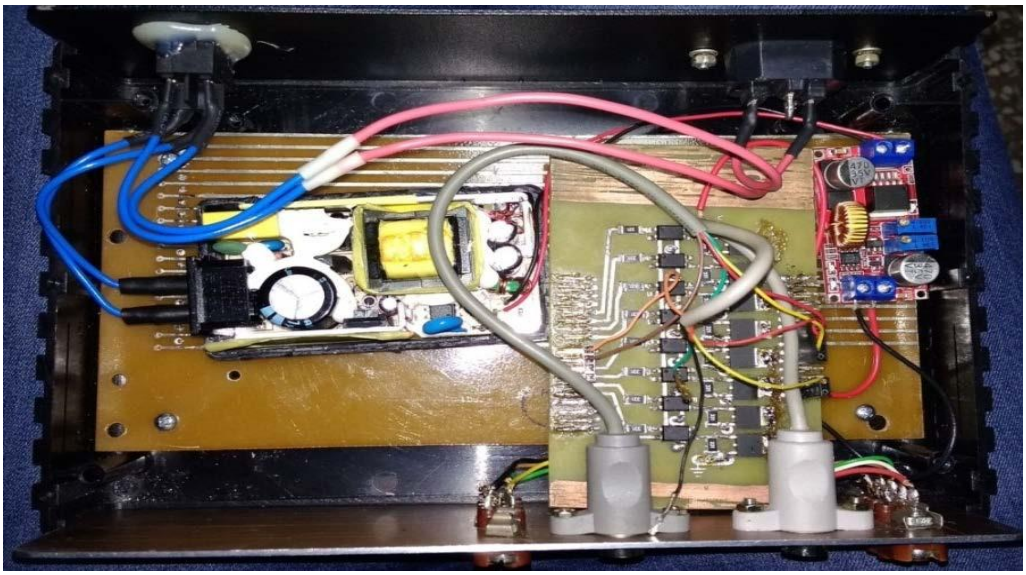


Рис.2.2. Фотографія блоку нового блоку керування кроковими двигунами (плата Arduino UNO знаходиться під платою ключів праворуч)

Для того, щоб написати код для мікроконтролеру та завантажити його ми використали середовище розробки Arduino IDE, яке містить редактор програмного коду, компілятор і модуль передачі коду в мікроконтролер. Дане середовище розробки підтримує мови C та C++ та їх бібліотеки, при цьому мають місце спеціальні правила структурування коду.

Загальний алгоритм коду, яким був прошитий мікроконтролер ATmega328P наступний: спочатку оголошуємо глобальні змінні, котрі будемо використовувати для запису даних з комп’ютера та подальшої їх обробки у коді. Більшість глобальних змінних – двовимірні масиви, що зумовлено наявністю двох крокових двигунів та необхідністю їх розмежування.

Далі, відповідно до стандарту написання коду в Arduino IDE виконуємо настройку входів/виходів(pin) плати, регістрів таймеру переривання, а також ініціюємо послідовне з’єднання з комп’ютером через емулятор COM-порту зі

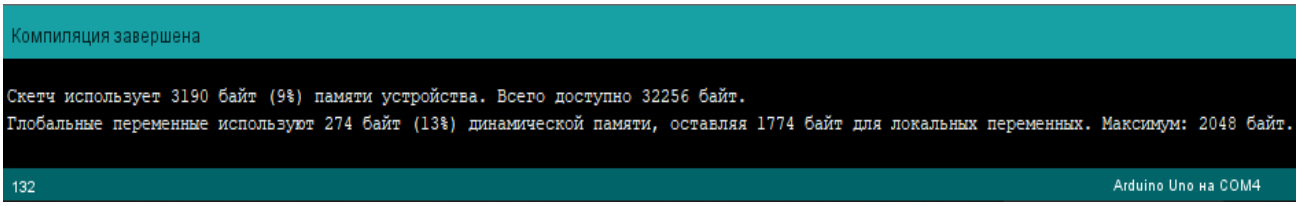
швидкістю 9600 біт/с в функції *setup()*. Ця функція викликається один раз при кожній подачі живлення або скидані Arduino.

Після цього прописуємо операції у тілі функції *loop()*. Ця функція буде виконуватись нескінченно циклічно доки плата працює, тобто до її відключення від живлення. Єдине і найлогічніше рішення, для реалізації керування кроковими двигунами при заданому часі на крок, яке не буде залежати від машинного часу – це винесення інструкцій виконання необхідної кількості кроків як функції, яка спрацьовує по перериванню, яке свою чергу буде залежати тільки від дільника тактової частоти плати(кварцовий резонатор 16МГц). Тобто у тілі функції *loop()* ми реалізуємо увімкнення та вимкнення таймеру переривання, прийняття вхідних даних з комп'ютера, а також їхню попередню обробку (перевірка кількості зчитаних байтів, вибір крокового двигуна, логіка вибору напрямку обертання ротору двигуна, приведення даних побітовим зсувом до потрібного виду для коректної подальшої обробки мікроконтролером) та виведення результату виконання роботи мікроконтролера(операція успішна, спрацювання кінцевика, помилка, тип помилки тощо).

Як було зазначено у параграфі вище, виконання кроку кроковим двигуном виконується по перериванню, а саме по співпадінню лічильника таймера з регістром співпадіння. Для реалізації цієї ідеї ми використали вбудований в Arduino Uno 16-бітний таймер/лічильник із модулем порівняння та дільником частоти. Інструкції по перемиканню фаз для виконання напівкроку (тобто подача сигналу з відповідних виводів плати), перевірка на спрацювання кінцевиків(перевірка логічного рівня на виводах плати, до яких підключені кінцевики), перевірка на виконання напівкрокового циклу двигуна закладені у функцію переривання *ISR(TIMER1_COMPA_vect)*, аргумент якої визначає номер лічильника та режим по співпадінню. Спочатку у функції переривання

виконується перевірка на спрацюванні кінцевиків з урахуванням напрямку обертання(для того щоб уникнути замикання кінцевика). Після цього виконується один напівкрок. Функція переривання буде повторюватись з частотою залежною від указаного в регістрі співпадіння *OCR1A* значення, яке записується з COM-порту та вказується у програмі вимірів користувачем - це значення фактично є часом виконання одного напівкроку. Лічильник буде вимкнено у випадках спрацювання кінцевиків, або успішного виконання зазначеної користувачем кількості напівкроків. У обох випадках з тіла функції *loop()* буде виведено інформацію щодо виконання команди у COM-порт, з ідентифікатором успішного виконання зазначеної кількості напівкроків, або спрацювання конкретного кінцевика. (Код програми наведений у Додатку 2)

Після написання коду його необхідно скомпілювати та завантажити в плату Arduino. Для цього спочатку вказуємо в Arduino IDE тип плати, який ми використовуємо: Инструменты -> Плата ->”Arduino Uno”, після цього вказуємо COM-порт, до якого підключена плата: Инструменты -> Порт ->COM4. Перед завантаженням у плату скомпілюємо і перевіримо на помилки. Для цього натискаємо Ctrl+R і отримуємо наступне повідомлення у вікні знизу:



Як бачимо помилок не має, і пам’яті пристрою ще достатньо для багатьох модифікацій і доповнень коду. Після цього натискання Ctrl+Shift+U завантажить код у мікроконтролер за допомогою програматора.

Далі представлена блок-схема принципу роботи коду(рис.2.3.):

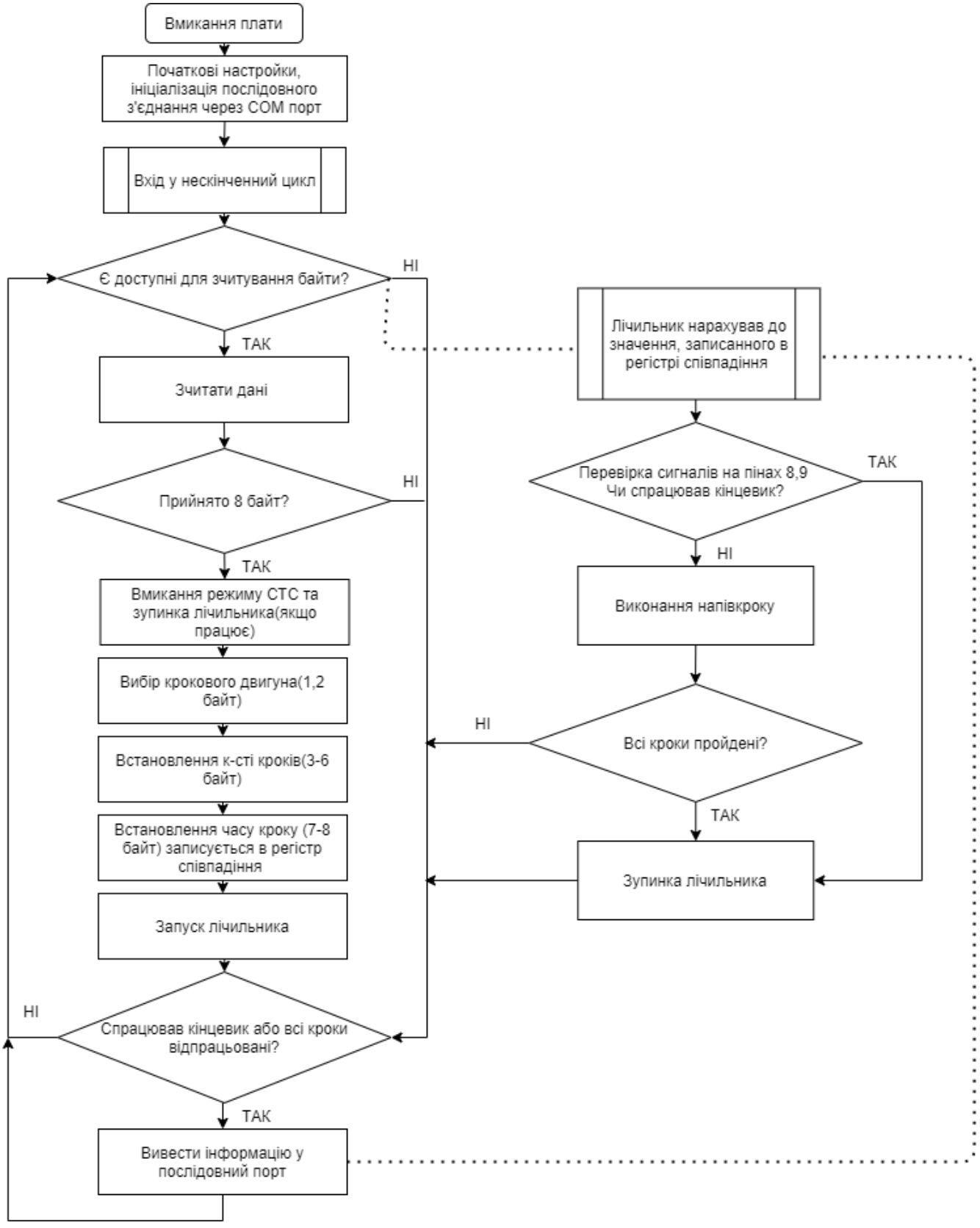


Рис.2.3. Блок-схема програми мікроконтролера

2.4. Програмний комплекс для обміну інформацією між платою і комп'ютером через COM-порт

Для реалізації надання команд з комп'ютера в мікроконтролер та прийняття даних з нього необхідно запрограмувати в комп'ютері потоки запису та зчитування даних з COM, функції відкриття та закриття послідовного порту, перевірки на помилки відкриття/закриття/зчитування/запису та їх тип, а також деякий масив, який буде виступати в ролі буфера для прийняття та обробки вхідних даних.

Все це було запрограмовано у одному заголовному файлі «хедері», для того, щоб у подальшому можна було користуватись ним у інших проектах, що працюють з Arduino, та легко підключати до необхідних програмних блоків інших проектів.

Основну для написання коду було взято з офіційного сайту розробника Arduino[12], при цьому до коду було внесено багато принципових змін: весь код реалізовано у одному заголовному файлі без використання класу, функції зчитування/запису реалізовано у вигляді потоків-функцій Windows API, додано функції відкриття/закриття послідовного порту тощо.

Код містить в собі необхідні змінні та функції для реалізації обміну через COM-порт, а саме: дескриптори порту, потоків зчитування/запису; структуру, яка виконує роль буферу для обробки інформації прийнятої з Arduino та містить флаг для перевірки завершення операційного циклу мікроконтролера, масиви буферів зчитування/запису для передачі інформації в плату; функції-потоки зчитування/запису, які відкриваються та закриваються після успішного відкриття/закриття COM відповідно; функція відкриття COM-порту, в якій вказуються параметри послідовного обміну(зчитування/запис, асинхронний режим тощо); ініціалізація системно визначеної структури DCB, члени якої

містять налаштування для послідовного обміну з периферійними пристроями(швидкість передачі, к-сть бітів в байті, задання перевірки парності, стоп-бітів тощо); поетапні перевірки на помилки та їх виведення у вікні.(Код програми наведений у Додатку 3)

2.5. Збільшення динамічного діапазону ФЕП

Для збільшення динамічного діапазону ФЕПа було зменшено сталу часу анодного кола ФЕПа. Для виконання цієї задачі потрібно було розібрати термоізолюючий корпус ФЕПа та перепаяти резистор навантаження в анодному колі пристрою (рис1.4). Але виконання цієї задачі даним чином викликало ряд проблем пов’язаних з порушенням термоізоляції та конструктиву у цілому. Тому було обрано менш ризиковий шлях: за допомогою нагрузки виходу ФЕПа, при підключенні операційного підсилювача з вхідним опором приблизно втричі меншим ніж R_H було зменшено постійну часу R_HC. При цьому зменшена амплітуда напруги вихідних імпульсів ФЕПа підсилюється операційним підсилювачем(ОП) у два рази, сам імпульс при цьому інвертується для зручнішої та більш логічної обробки у подальшому.

Для виконання цієї задачі було спаяно схему інвертуючого ОП с коефіцієнтом підсилення к=2 та підключеним потенціометром для установки нулю. У якості ОП було обрано мікросхему AD645.

Після цього було написано програму для відображення кількості точок, виміряних АЦП(PCI9112) за один фотоімпульс. Виявилось, що на один імпульс у середньому припадає 7 точок, з урахуванням того, що частота опитувань плати 110кГц це означає, що інтервал одного імпульсу ФЕПа дорівнює 56 мкс, при попередньому підсилювачі інтервал одного імпульсу сягав близько 200 мкс. Тобто тривалість імпульсу зменшилась приблизно у 4 рази. Це має дати більшу

точність вимірювань за великих інтенсивностей, адже зменьшилась ймовірність накладання двох фотонів у один імпульс на виході. Також потрібно зазначити, що даної кількості точок на імпульс цілком достатньо для використання алгоритму підрахунку фотонів цієї установки. Алгоритм полягає в тому, що виявлення імпульсу відбувається за наявністю точки перегину на послідовності 5 вимірів. Тобто для обраної точки перевіряється умова, відповідно до якої по дві сусідні точки ліворуч і праворуч повинні мати значення, яке буде меньше від значення заданої точки. Якщо така умова виконується, то обрана точка вважається вершиною імпульсу, імпульс ідентифікується[11]. Також для збільшення точності вимірів, точка, яка пройшла перевірку на пік додатково перевіряється на величину амплітуди, якщо її значення більше ніж значення шумового сигналу, то імпульс фіксується.

Окрім цього критерій переповнення АЦП, границею вхідного сигналу якого є 10В став мати більш наочне фізичне значення, адже після інвертування сигналу ФЕПа нуль сигналу відповідає мінімальним значенням напруги, а не коливається в межах максимального як було до цього.

2.6. Автономна система термостабілізації ФЕП

Попередній варіант системи термостабілізації був підв’язанний під програму в комп’ютері, це призводило до негативних результатів у виді переохолодження ФЕПа, у разі зависання комп’ютера або програми. Тому було прийнято рішення реалізувати подібну до старої систему термостабілізації, але зробити її повністю автономною. Для цього було обрано цифровий терморегулятор W1209 зі зручним інтерфейсом, який здатен підтримувати температуру стабільно з достатньою точністю 0.1°C. W1209 має реле, яке

вмикається та вимикається в залежності від обраного режиму роботи(охолодження/нагрів).Обрання одного чи іншого режиму не принципове.

Блок-схема термостабілізації наведена на рис.2.4.

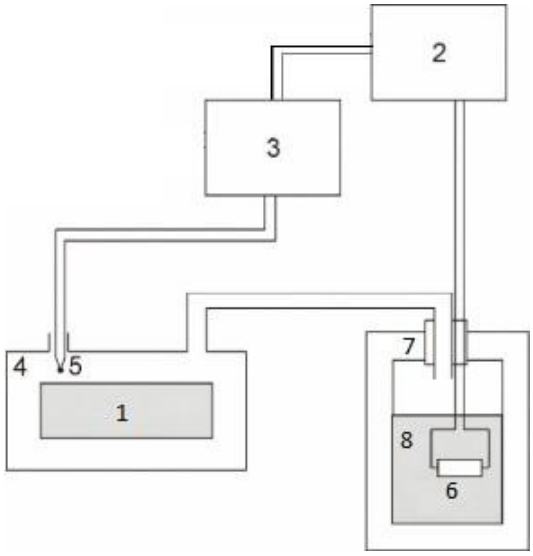


Рис.2.4. Блок-схема термостабілізації ФЕПа:

1-ФЕП;2-Блок живлення резистора;3-W1209;4-термоізолюючий корпус;5-датчик температури;6-нагріваючий резистор;7-посудина Дюара;8-рідкий азот.

Датчик терморегулятора розміщується у спеціальному вікні термоізолюючого корпусу ФЕПа, де він знімає температуру, при перевищені заданого порогу температури на 0.1°C реле терморегулятора замикає контакти, в результаті чого з блоку джерела живлення резистора, зануреного в посудину Дюара з рідким азотом, поступає струм, який призводить до виділення Джоулівського тепла на резисторі та випаровування рідкого азоту. Холодні пари азоту потрапляють до термоізолюючого корпусу, внаслідок чого відбувається охолодження ФЕПа.

Конструктивно отвір термоізолюючого корпусу, з якого надходять пари азоту, знаходиться над резисторами подільника напруги ФЕПа та опору навантаження.

2.7. Оптимізація програми

Оптимізація програм проекту була виконана мовою програмування C++ у середовищі Builder C++, в якому була написана до цього.

Основною задачею оптимізації було перепрограмувати старі алгоритми керування кроковими двигунами та підрахунку фотонів. Проект буд написаний модульно, з використанням класів та власних просторів імен, що значно спростило заміну старих блоків коду. Також було необхідно видалити старий інтерфейс ініціалізації плати ET3401 та запрограмувати новий інтерфейс для відкриття та закриття COM-порту з оптимізацією, для уникнення перекривання функцій відкриття/закриття, що можуть бути викликані з різних місць в проекті, так як він є досить великим. Вигляд інтерфейсу наведено на рис. 2.5.

Для заміни модуля керування кроковими двигунами з проекту було повністю видалено дві програми з їхніми заголовними файлами у результаті того, що їхні функції були замінені Arduino. Було повністю перепрограмовано функції, які відповідали за виконання наступних операцій: встановлення кроку кроковим двигуном та перевірка на спрацювання кінцевиків, встановлення довжини хвилі(яку виділяє монохроматор) та встановлення нулю монохроматора. Для коректної роботи комп’ютера з мікроконтролером до головного файлу проекту було додано дві функції для відправлення команд- *SendCommand(int.int.int)* та очікування відповіді від мікроконтролера – *COM_ReadCheck()*. Ці функції визиваються у всіх частинах проекту при надсиланні команд до мікроконтролеру, та виконують перевірку відповіді з боку мікроконтролера.(Код програми наведений у Додатку 4)

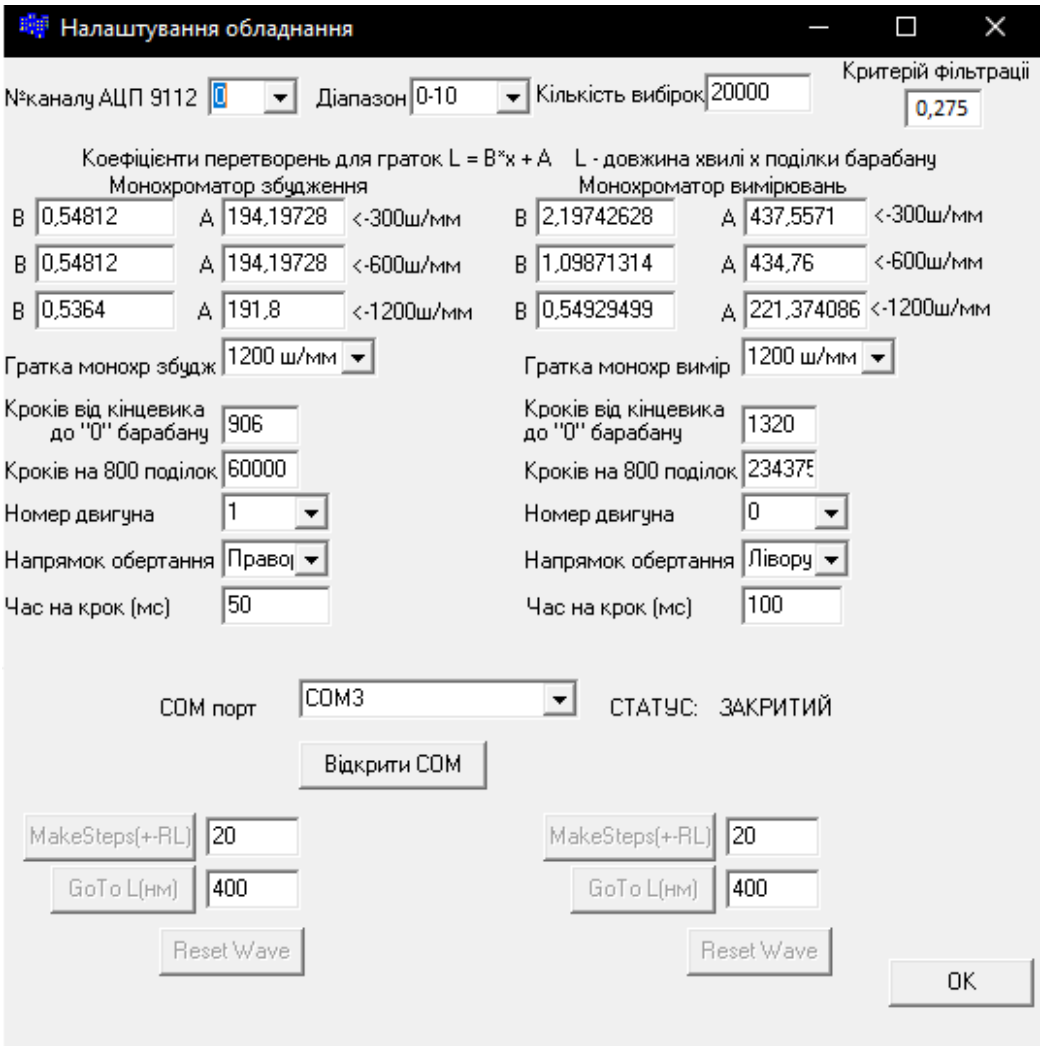


Рис. 2.5. Інтерфейс для налаштування обладнання

Після заміни підсилювача, який використовується для зняття сигналу з ФЕПа на інвертуючий. Необхідно було перепрограмувати алгоритми підрахунку імпульсів фотонів та вимірювання середньої напруги, знятої з R_H ФЕПа. Після того як це було виконано і результати вимірювань показали задовільні результати, було виміряно новий критерій відсікання темнового фотоструму, для цього були виміряні шумові сигнали при повністю закритих щілинах монохроматора. Ці виміри показали, що рівень темнового фотоструму відповідає близько 0,275В знятих АЦП. Після отримання цього параметру було додано функцію вибору критерія відсікання імпульсів менше заданого

користувачем значення амплітуди (у графі «Налаштування»-> «Обладнання»). Також у графі «Налаштування» було додано опцію «Налаштування положення ФЕП», яка відображає залежність кількості підрахованих фотонів у часі. Використання цієї програми дозволяє перевірити стабільність випромінювання джерела, та налаштувати оптимальне положення ФЕПа., поки вона працює у реальному часі(рис.2.6).

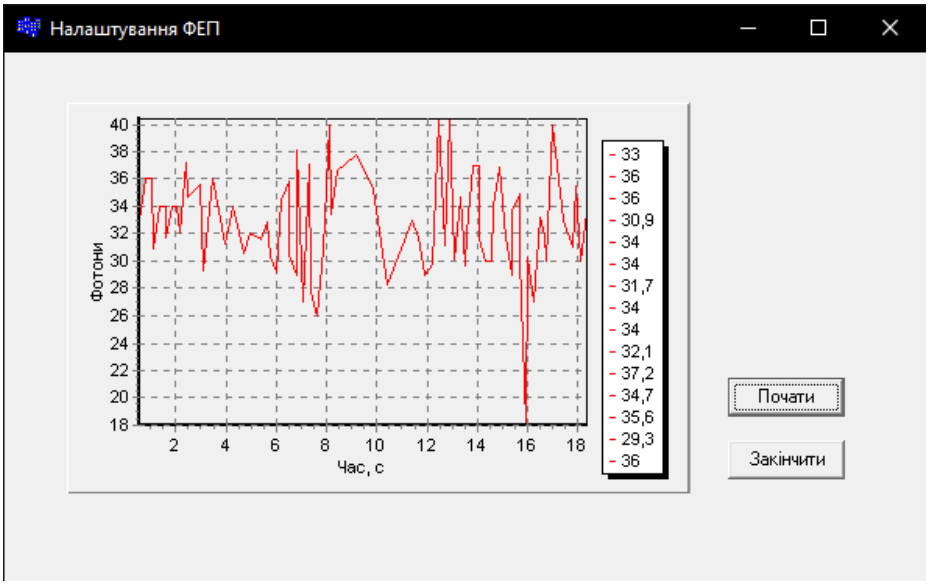


Рис.2.6. Програма для перевірки стабільності роботи джерела

2.8. Апробація установки

Для апробації установки було обрано білий люмінофорний світлодіод, на основі синього світлодіоду, покритого люмінофором, що у результаті фотолюмінесценції перетворює первинну частину випромінювання у світло з відносно широким спектром з максимумом у зелено-жовтій області видимого діапазону. Світлодіод живиться від USB підключеного до комп'ютера. Для виконання вимірів був використаний один монохроматор, щоб виконати виміри по всьому спектру. При вимірюванні була використана дифракційна гратка 600

шт./нм, з робочою областю 430-1250мм. Виміри проводились за однакових умов: діапазон вимірювань - 435-730нм; однакова оптична система; крок вимірювання 0.5нм, 20 усереднень на вимір, але при цьому змінювався розмір вхідної щілини монохроматора, для забезпечення різних інтенсивностей і перевірки динамічного діапазону(спотворення спектрів за більших інтенсивностей). Інтенсивність спектрів за кольорами: червоний>оранжевий.>синій.

Невіднормовані виміряні спектри зображені на рис.2.7:

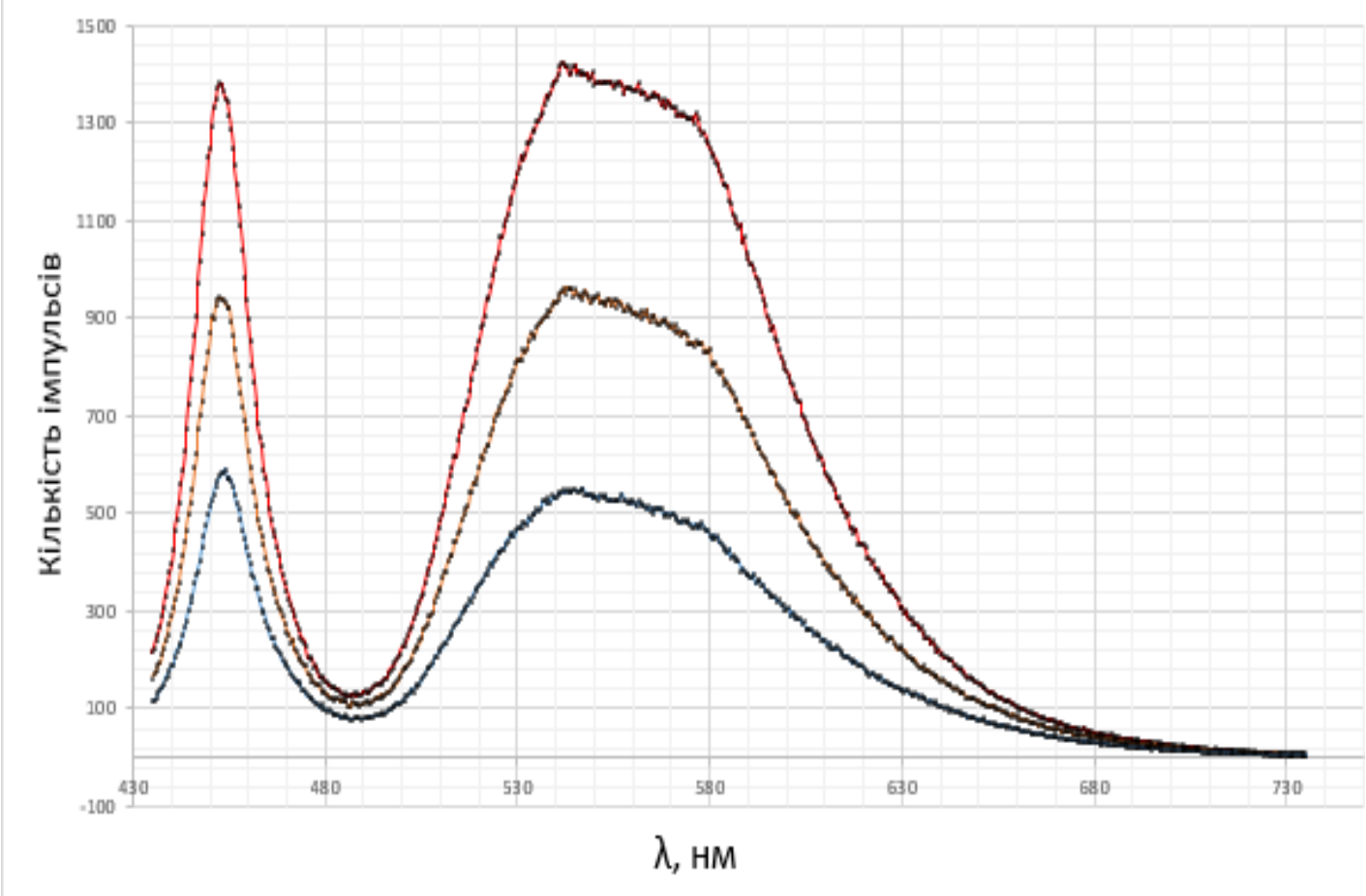


Рис.2.7. Невіднормовані виміряні спектри білого світлодіоду

Як бачимо положення піків та форма спектрів при різних інтенсивностях співпадає з великою точністю. З графіку видно, що спектр з найбільшою інтенсивністю виходить на сигнал близько 1500 підрахованих імпульсів ФЕПа, при цьому можна бачити чітку форму спектру, тоді як з використанням старого підсилювача сигнал виходив на нелінійність при інтенсивностях близько 1000 імпульсів, тобто можна зробити висновок, що спроба збільшити динамічний діапазон ФЕПа виявилась вдалою. Але для кращого порівняння та перевірки на спотворення спектру при великих інтенсивностях необхідно віднормувати кожен

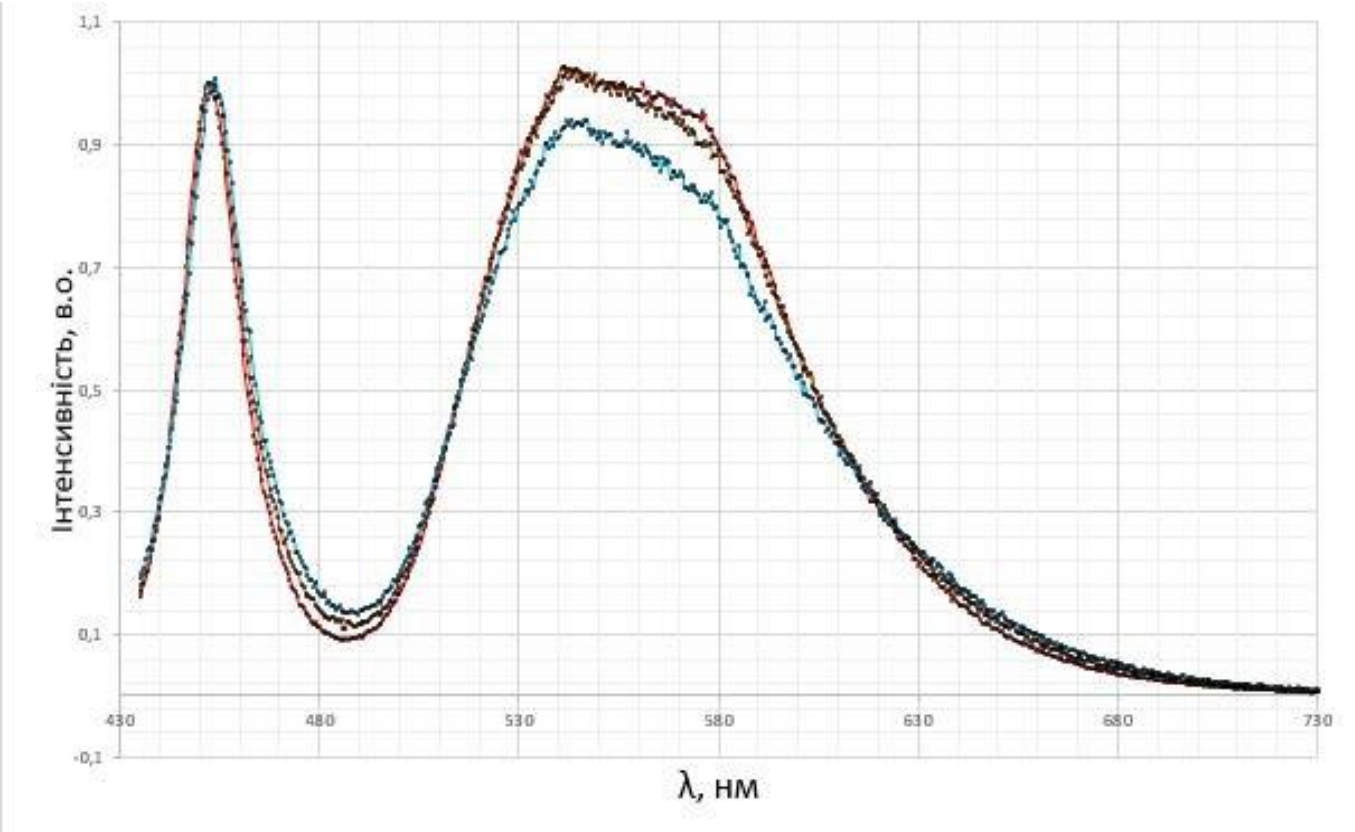


Рис.2.8. Віднормовані на максимум спектри білого світлодіоду:

спектр відносно його максимуму і зобразити їх також на одному графіку:

Як бачимо деяка деформація спектру все ж таки присутня. По-перше зі зменшенням загальної інтенсивності вимірювання співвідношення інтенсивностей піків змінюється. Пік у жовто-зеленій області видимого діапазону стає меншим за пік у синьому. Також при більших інтенсивностях збільшується провал у блакитно-зеленій області спектру. З чим пов’язана така поведінка вимірювальної системи сказати важко і це потребує подальшого розгляду(перевірка стабільності джерела, врахування спектральної чутливості установки тощо).

Висновки

- модернізовано блоки: керування кроковими двигунами, вимірювання, системи термостабілізації;
- запрограмовано мікроконтролер для автоматизації керування кроковими двигунами;
- зменшено тривалість та інвертовано вихідні імпульси ФЕПа для збільшення динамічного діапазону;
- оптимізовано програмний код проекту для роботи з новим блоком керування кроковими двигунами, добавлено функцію перевірки стабільності сигналу з ФЕПа у реальному часі;
- проведено апробацію установки, що показала коректну роботу крокових двигунів та нового підсилювача сигналу ФЕПа;

Перелік посилань

1. А.В. Емельянов, А.Н. Шилин. Шаговые двигатели:учебное пособие.- Волгоград,2005. с.7-13
2. Анисимова И.И., Глуховской Б.М., Фотоэлектронные умножители. М.: Сов.радио, 1974. — 376 с.
3. Пейсахсон И.В. Оптика спектральных приборов. Л.: Машиностроение, 1975. —312 с.
4. Тарасов К.И. Спектральные приборы. Л.: Машиностроение, 1977 — 367 с.
5. Young A.T. Photomultipliers, in: Methods of Experimental Physics. V.12, Part A. NY.: Acad Press. 1974.
6. В.Корнилов и А.Крылов Четирьох канальный спектроанализатор для исследования яркости звезд. Астрон.Ж., 1991, Т. 67, С. 173-181.
7. https://voltiq.ru/wiki/arduino-uno-review
8. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
9. Ключев А.О., Ковязина Д.Р., Петров Е.В., Платунов А.Е. Интерфейсы периферийных устройств. – СПб.: СПбГУ ИТМО, 2010. с.72-75.
10. В.С.Баран, Г.Г.Власюк, Ю.О.Оникієнко, О.І.Смоленська., Основи мікропроцесорної техніки.– Київ : КПІ ім. Ігоря Сікорського, 2019. с.33-36
11. Вісник київського університету, Серія: Фізико-математичні науки Випуск №4 Бунак С.В., Ільченко В.В.,Окоронко О.М.,Шкавро А.Г. «Проста

реалізація методу підрахунку фотонів для дослідження спектрів поверхневобар'єрних наногетероструктур» ст.249

12. https://playground.arduino.cc/Interfacing/CPPWindows/

Додатки

Додаток 1: Схема блоку керування кроковими двигунами(рис.3.1.) та плати ключів, яка підключається до виводів Arduino Uno(рис.3.2.) для реалізації напівкрокового режиму.

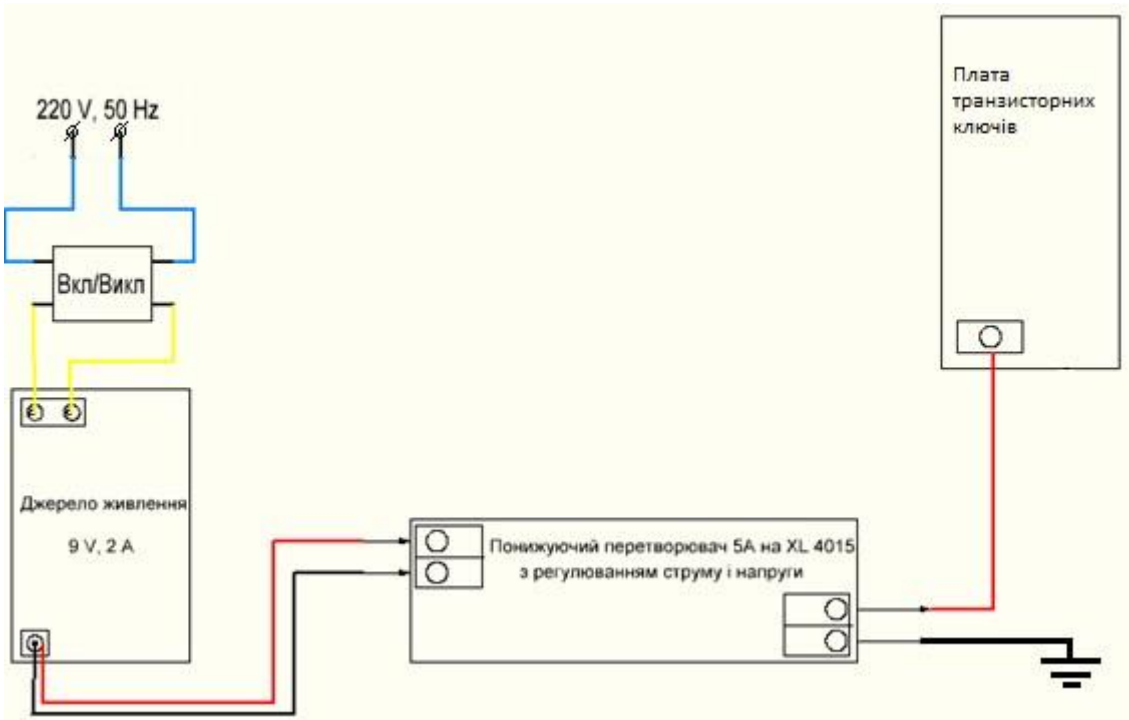


Рис.3.1. Схема блоку керування кроковими двигунами

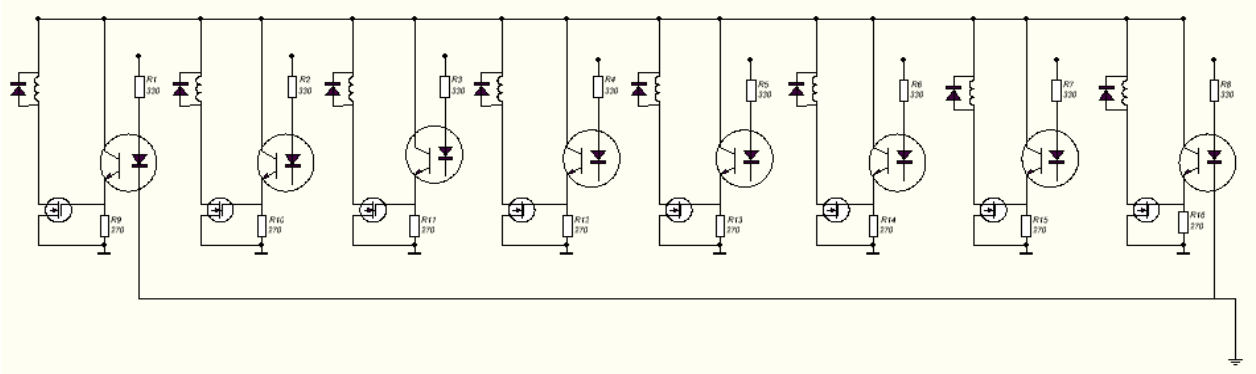


Рис.3.2. Схема плати ключів для напівкрокового режиму керування двигунами

Додаток 2: Код програми, якою був прошитий мікроконтролер ATmega328P

```
int i[2]={0,0};

long real_step[2]={0,0};// поточне значення кроків

long number_of_step[2]; // кількість кроків

char inc[2]={0,0};//int змінна що визначає напрямок руху двигуна signed byte

int inform;//1 - спрацювання кінцевику першого двигуна; 2 - другого; 10 - позиція встановлена

int interval[2];//тривалість кроку в папугах(1 папуга(У.О) орієнтовно 0.1мс)

byte message_in[8], message_out[8];//масиви в яких передаються і приймаються дані на/з компа 8 байт

void setup()

{

 DDRB = B11111100;// устанавливаем режим работы вывода, как "выход" и 2 бита на вход

 PORTB = B00000011;// 2 останні біта "підтягнуті" резистором до напруги живлення МК

 DDRD = B11110010;//4 виходи(двигун) 2 входи(кінцевики) 1вихід(TX) 1 вхід(RX)

 PORTD = B00001111;

 TCCR1A=B00000000;

 TCCR1C=B00000000;

 TIMSK1=B00000010;

// real_step=0;

Serial.begin(9600);

}

ISR(TIMER1_COMPA_vect)// описується переривання, по співпадінню регістра таймера

{

if(((~PINB&B00000010)&&(inc[message_in[1]]&B10000000))||((~PINB& B00000001)&&(inc[message_in[1]] >0)))

 {

  TCCR1B=B00001000;

 if (~PINB & B00000001)

 {

  inform = 1;
```

```
 }
 if (~PINB & B00000010)
 {
  inform = 2;
 }
 }//перевірка на спрацювання кінцевиків
 else
 {
 i[message_in[1]]+=inc[message_in[1]];///прирощування кроку з урахуванням напрямку вн.циклічне

real_step[message_in[1]]+=inc[message_in[1]];//прирощування кроку з урахуванням напрямку загальне////

 if(i[message_in[1]]==-1) i[message_in[1]]=7;//перевірка на завершення напівкрокового циклу двигуна
 if(i[message_in[1]]==8) i[message_in[1]]=0;//перевірка на завершення напівкрокового циклу двигуна
 if (message_in[1]==0)
 {
  switch (i[0])
 {
   case 0: PORTB=B00100011;break;
   case 1: PORTB=B00110011;break;
   case 2: PORTB=B00010011;break;
   case 3: PORTB=B00011011;break;
   case 4: PORTB=B00001011;break;
   case 5: PORTB=B00001111;break;
   case 6: PORTB=B00000111;break;
   case 7: PORTB=B00100111;break;
 }// виконання кроку двигуна 0
 }
  if (message_in[1]==1)
 {
  switch (i[1])
 {
   case 0: PORTD=B10001111;break;
   case 1: PORTD=B11001111;break;
   case 2: PORTD=B01001111;break;
   case 3: PORTD=B01101111;break;
   case 4: PORTD=B00101111;break;
   case 5: PORTD=B00111111;break;
   case 6: PORTD=B00011111;break;
   case 7: PORTD=B10011111;break;
 }// виконання кроку двигуна 1
 }
  if ( number_of_step[message_in[1]]==real_step[message_in[1]])
 {
 TCCR1B=B00001000;//зупинка лічильника
 inform =10;
 } //перевірка чи всі кроки пройдені
 }
}
 void loop ()
 {
    if (Serial.available()>0)
    {
     message_in[q] = Serial.read();
     q++;
```

```
    }// читання даних з COM - порта
   if(q==8)
   {

if((message_in[0]==68)&&((message_in[1]==0)||(message_in[1]==1)))
    {
     TCCR1B=B00001000;///////////////зупинка лічильника
     number_of_step[message_in[1]]=0;

number_of_step[message_in[1]]+=message_in[5];

number_of_step[message_in[1]]=(number_of_step[message_in[1]]<<8) + message_in[4];

number_of_step[message_in[1]]=(number_of_step[message_in[1]]<<8) + message_in[3];

number_of_step[message_in[1]]=(number_of_step[message_in[1]]<<8) + message_in[2];
     interval[message_in[1]]=0;
     interval[message_in[1]]+=message_in[7];

interval[message_in[1]]=(interval[message_in[1]]<<8) +message_in[6];
     OCR1A = interval[message_in[1]] * 5;
     inc[message_in[1]]=( number_of_step[message_in[1]]>=0)?(1):(-1);
     real_step[message_in[1]]=0;
     if ( number_of_step[message_in[1]] !=0 )
     TCCR1B=B00001100;  // запуск лічильника з дільником clk/256
    }
     q=0;
    }// обробка інформації
   if (inform >0)
   {
    if (inform==10)
    {
     message_out[0]=68;
     message_out[1]=message_in[1];
     message_out[5]=0;
     message_out[4]=0;
     message_out[3]=0;
     message_out[2]=0;
     message_out[6]=79;//O
     message_out[7]=75;//K
    }
    if (inform==1)
    {
  message_out[0]=68;
message_out[1]=message_in[1];
message_out[5]=real_step[message_in[1]];
message_out[4]=real_step[message_in[1]]>>8;

message_out[3]=real_step[message_in[1]]>>16;

message_out[2]=real_step[message_in[1]]>>24;
message_out[6]=76; //спрацював кінцевик 1-го двигуна
message_out[7]=0;
    }
if (inform==2)
    {
```

```
message_out[0]=68;
message_out[1]=message_in[1];
message_out[5]=real_step[message_in[1]];
message_out[4]=real_step[message_in[1]]>>8;
message_out[3]=real_step[message_in[1]]>>16;
message_out[2]=real_step[message_in[1]]>>24;
message_out[6]=82; //спрацював кінцевик 2-го двигуна
message_out[7]=0;
    }
   Serial.write(message_out,8);//виведення інформації у COM-порт
        inform = 0;
   }
}
```

Додаток 3: Код програми для забезпечення обміну інформації між комп'ютером та мікроконтролером з боку ПК

```
#define BUFSIZE 128 //ёмкость буфера
DWORD WINAPI ReadThread(LPVOID);
DWORD WINAPI WriteThread(LPVOID);
int  COMOpen();
int COMClose();

unsigned char bufrd[BUFSIZE], bufwr[BUFSIZE]; //приёмный и передающий буферы

int ComOpen;
HANDLE COMport=NULL;
OVERLAPPED overlappedwr;
OVERLAPPED  overlappedrd;
HANDLE reader;
HANDLE writer;

struct TrnsVar{
int flag;
unsigned char * message;
} TransferVar;

//главная функция потока, реализует приём байтов из COM-порта
DWORD WINAPI ReadThread(LPVOID)
{
 COMSTAT comstat;        //структура текущего состояния порта, в данной программе
                         //используется для определения количества принятых в порт байтов
 DWORD btr, temp, mask, signal;
 int counter,tempz; //переменная temp используется в качестве заглушки

 overlappedrd.hEvent = CreateEvent(NULL, true, true, NULL); // создать сигнальный объект-событие для асинхронных операций
 SetCommMask(COMport, EV_RXCHAR); //установить маску на срабатывание по событию приёма байта в порт
 while(1) //пока поток не будет прерван, выполняем цикл
```

```
 {
  WaitCommEvent(COMport, &mask, &overlappedrd); // ожидать события приёма байта
// (это и есть перекрываемая операция)
  signal = WaitForSingleObject(overlappedrd.hEvent, INFINITE); //приостановить поток до прихода байта
  if(signal == WAIT_OBJECT_0)  //если событие прихода байта произошло
   {
    //проверяем, успешно ли завершилась перекрываемая операция WaitCommEvent
    if(GetOverlappedResult(COMport, &overlappedrd, &temp, true))
     if((mask & EV_RXCHAR)!=0) //если произошло именно событие прихода байта
      {
       ClearCommError(COMport, &temp, &comstat); //нужно заполнить структуру COMSTAT
       btr = comstat.cbInQue; //и получить из неё количество принятых байтов
       if(btr) //если действительно есть байты для чтения
        {
         ReadFile(COMport, bufrd, btr, &temp, &overlappedrd); //прочитать байты из порта в буфер программы

         TransferVar.flag=btr;
         TransferVar.message = bufrd;

        }
      }
   }
 }
}

//-----------------------------------------------
DWORD WINAPI WriteThread(LPVOID)
{
 DWORD temp, signal;  //temp - переменная-заглушка

 overlappedwr.hEvent = CreateEvent(NULL, true, true, NULL);  //создать событие
 while(1)
  {
   //записать байты в порт (перекрываемая операция!)
   WriteFile(COMport, bufwr, strlen(bufwr), &temp, &overlappedwr);

   //приостановить поток, пока не завершится перекрываемая операция WriteFile
   signal = WaitForSingleObject(overlappedwr.hEvent, INFINITE);

   //если операция завершилась успешно
   if((signal == WAIT_OBJECT_0) && (GetOverlappedResult(COMport, &overlappedwr, &temp, true)))
    {
     //вывести сообщение об этом в строке состояния
```

```
    Form1->Memo1->Lines->Add("Передача прошла успешно");
      }
   //иначе вывести в строке состояния сообщение об ошибке
   else
   {                  // MessageBox
   Form1->Memo1->Lines->Add("Ошибка передачи");
   }

   SuspendThread(writer);
  }

}

 //-------------------------------------------------------
----------
//функция открытия и инициализации порта
int COMOpen(String portname) //1 вже відкритий 0 вдало відкритий -1,-2,-3,-4 помилки
{
        //имя порта (например, "COM1", "COM2" и т.д.)
 DCB dcb;                //структура для общей инициализации порта DCB
 COMMTIMEOUTS timeouts;
        //структура для установки таймаутов

 //portname = Form1->ComboBox1->Text;
        //получить имя выбранного порта

  if(COMport==NULL)//!=NULL)
  {
 //открыть порт, для асинхронных операций обязательно нужно указать флаг FILE_FLAG_OVERLAPPED
 COMport = CreateFile(portname.c_str()
 /*"COM1"*/,GENERIC_READ | GENERIC_WRITE, 0,NULL, OPEN_EXISTING,
 FILE_FLAG_OVERLAPPED, NULL);
 //здесь:
 // - portname.c_str() - имя порта в качестве имени файла,
 // c_str() преобразует строку типа String в строку в виде массива типа char, иначе функция не примет
 // - GENERIC_READ | GENERIC_WRITE - доступ к порту на чтение/запись
 // - 0 - порт не может быть общедоступным (shared)
 // - NULL - дескриптор порта не наследуется, используется дескриптор безопасности по умолчанию
 // - OPEN_EXISTING - порт должен открываться как уже существующий файл
 // - FILE_FLAG_OVERLAPPED - этот флаг указывает на использование асинхронных операций
 // - NULL - указатель на файл шаблона не используется при работе с портами

 if(COMport == INVALID_HANDLE_VALUE) //если ошибка открытия порта
  {
 //                Form1->Memo1->Lines->Add(TimeToStr(Now())+ " : Не удалось открыть порт");
```

```
COMport = NULL;
  return -1;
 }
 //инициализация порта

 dcb.DCBlength = sizeof(DCB);       //в первое поле структуры DCB необходимо занести её длину,
                                    //она будет использоваться функциями настройки порта
                                    //для контроля корректности структуры

 //считать структуру DCB из порта
 if(!GetCommState(COMport, &dcb))
 {
  //если не удалось - закрыть порт и вывести сообщение об ошибке в строке состояния
  COMClose();
  Form1->Memo1->Lines->Add("Не удалось считать DCB");
  COMport = NULL;
  return -2;
 }

 //инициализация структуры DCB
dcb.BaudRate = 9600; //задаём скорость передачи мин-9600,макс-115200 бод
dcb.fDtrControl = DTR_CONTROL_ENABLE;
//отключаем использование линии DTR
dcb.fRtsControl = RTS_CONTROL_ENABLE;
//отключаем использование линии RTS
dcb.ByteSize = 8;                  //задаём 8 бит в байте
dcb.Parity = 0;                    //отключаем проверку чётности
dcb.StopBits = 0;                  //задаём один стоп-бит
 //загрузить структуру DCB в порт
 if(!SetCommState(COMport, &dcb))
 {
  //если не удалось - закрыть порт и вывести сообщение об ошибке в строке состояния
  COMClose();
  Form1->Memo1->Lines->Add("Не удалось установить DCB");
  COMport = NULL;
  return -3;
 }

 //установить таймауты
 timeouts.ReadIntervalTimeout = 0;
        //таймаут между двумя символами
 timeouts.ReadTotalTimeoutMultiplier = 0;
        //общий таймаут операции чтения
 timeouts.ReadTotalTimeoutConstant = 0;
//константа для общего таймаута операции чтения
 timeouts.WriteTotalTimeoutMultiplier = 0;
//общий таймаут операции записи
 timeouts.WriteTotalTimeoutConstant = 0;
//константа для общего таймаута операции записи
```

```
 //записать структуру таймаутов в порт
 //если не удалось - закрыть порт и вывести сообщение об ошибке в строке состояния
 if(!SetCommTimeouts(COMport, &timeouts))
  {
   COMClose();
   Form1->Memo1->Lines->Add("Не удалось установить тайм-ауты");
   COMport = NULL;
   return -4;
  }

PurgeComm(COMport, PURGE_RXCLEAR);  //очистить принимающий буфер порта

//создаём поток чтения, который сразу начнёт выполняться (предпоследний параметр = 0)
reader = CreateThread(NULL, 0, ReadThread, NULL, 0, NULL);
//создаём поток записи в остановленном состоянии (предпоследний параметр = CREATE_SUSPENDED)
writer = CreateThread(NULL, 0, WriteThread, NULL, CREATE_SUSPENDED, NULL);
return 0;
}
else
{
return 1;
}
}

//-------------------------------------------------------------------
//функция закрытия порта
int COMClose()
{
if(COMport!=NULL)
{
//Примечание: так как при прерывании потоков, созданных с помощью функций WinAPI, функцией TerminateThread
//  поток может быть прерван жёстко, в любом месте своего выполнения, то освобождать дескриптор
//  сигнального объекта-события, находящегося в структуре типа OVERLAPPED, связанной с потоком,
//  следует не внутри кода потока, а отдельно, после вызова функции TerminateThread.
//  После чего нужно освободить и сам дескриптор потока.

 //если поток записи работает, завершить его; проверка if(writer) обязательна, иначе возникают ошибки
if(writer)
  {
TerminateThread(writer,0);
 CloseHandle(overlappedwr.hEvent);// нужно закрыть объект-событие
  CloseHandle(writer);
  }
```

```
 //если поток чтения работает, завершить его; проверка if(reader) обязательна, иначе возникают ошибки


 if(reader)
  {TerminateThread(reader,0);
   CloseHandle(overlappedrd.hEvent);
         //нужно закрыть объект-событие
   CloseHandle(reader);
  }
 CloseHandle(COMport);  //закрыть порт
 COMport=0;            //обнулить переменную для дескриптора порта
 return 0;
 }
 else
 {
 return -1;
 }
}
```

Додаток 4: Код оптимізованого модулю проекту для виконання вимірювів

```
#include "Monochromator.h"
#include "Unit1.h"
#include <math.h>
#include <vcl.h>
#include "dask.h"
#include <dos.h>
#include <stdio.h>

extern void SendCommand(int,int,int);
extern int COM_ReadCheck();
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
int        Foto::Monochromator::MoveStep(int step,int delay,int RB)
{
 if(step!=0)
      {
if((StepMotorNumber!=0)&&(StepMotorNumber!=1))
            {
            Form1->Memo1->Lines->Add(TimeToStr(Now())+":          Помилка ініціалізації двигунів!");
            return -1;
            }

(StepMotorNumber==0)?(SendCommand(68,step*RB,delay)):(SendCommand(324,step*RB,delay));

      int check = COM_ReadCheck();
      switch(check){

      case 1: return 0;

      case (-1):
      WaveLength=-1;
```

```
        return -2;


        case 0: return -1;
        }
        }
 return 0;
}
//-------------------------------------------------------------------


//-------------------------------------------------------------------

Foto::Monochromator::Monochromator(double A300,double B300,double A600,double B600,double A1200,double B1200,int gratka){


Gratka300.koeffA=A300;
Gratka300.koeffB=B300;
Gratka600.koeffA=A600;
Gratka600.koeffB=B600;
Gratka1200.koeffA=A1200;
Gratka1200.koeffB=B1200;
Gratka=gratka;
InitialWave= (Gratka==300)?(Gratka300.koeffA):((Gratka==600)?(Gratka600.koeffA):((Gratka==1200)?(Gratka1200.koeffA):(0)));


StepCount=0;
WaveLength=-1;


}

//-------------------------------------------------------------------

void Foto::Monochromator::SetStepParams(int StepMotorNumber1,int InitSteps,int StepsP800,int direction){
RotateDirection=direction;
InitialStep= InitSteps;
StepsPer800=StepsP800;
StepMotorNumber =  StepMotorNumber1;
if(Gratka==300)
{
LPerStep=(800.0*Gratka300.koeffB)/StepsPer800;
InitialWave=Gratka300.koeffA;
}


if(Gratka==600)
{
LPerStep=(800.0*Gratka600.koeffB)/StepsPer800;
InitialWave=Gratka600.koeffA;
}


if(Gratka==1200)
{
LPerStep=(800.0*Gratka1200.koeffB)/StepsPer800;
InitialWave=Gratka1200.koeffA;
}


}
//-------------------------------------------------------------------
```

```
int Foto::Monochromator::SetWave(double lambda, int ttime){


int DStep,Step;
if (WaveLength==-1)
{
Form1->Memo1->Lines->Add(TimeToStr(Now())+": Не проведена початкова ініціалізація");
return -1;
}


Step = ((int)((lambda-InitialWave)/LPerStep*10.0)%10 >=5 )?((int)((lambda-InitialWave)/LPerStep)+1):((int)((lambda-InitialWave)/LPerStep));
DStep=Step-StepCount;
if (DStep<0)
{
if(MoveStep((DStep-300),ttime,RotateDirection)<0) // DStep кроків time мс на крок
    {
    Form1->Memo1->Lines->Add(TimeToStr(Now())+": Помилка при встановлені довжини хвилі!");
    WaveLength=-1;
    return -2;
    }
if(MoveStep(300,ttime,RotateDirection)<0) //вибираємо люфт системи(редуктор+барабан+гратка)
    {
    Form1->Memo1->Lines->Add(TimeToStr(Now())+": Помилка при встановлені довжини хвилі!");
    WaveLength=-1;
    return -2;
    }


}
else
{
if(MoveStep(DStep,ttime,RotateDirection)<0)
    {
    Form1->Memo1->Lines->Add(TimeToStr(Now())+": Помилка при встановлені довжини хвилі!");
    WaveLength=-1;
    return -2;
    }
}


WaveLength=lambda;
StepCount=Step;
return 0;
}
//-------------------------------------------------------------

int Foto::Monochromator::GetGratka(){//показуэ яка гратка встановлена 300 600 1200
return  Gratka;
}
//-------------------------------------------------------------
```

```
void
Foto::Monochromator::SetGratkasParams(dou
ble A300,double B300,double A600,double
B600,double A1200,double B1200)

{

Gratka300.koeffA=A300;

Gratka300.koeffB=B300;

Gratka600.koeffA=A600;

Gratka600.koeffB=B600;

Gratka1200.koeffA=A1200;

Gratka1200.koeffB=B1200;


if(Gratka==300)

{

LPerStep=(800.0*Gratka300.koeffB)/StepsPe
r800;

InitialWave=Gratka300.koeffA;

}


if(Gratka==600)

{

LPerStep=(800.0*Gratka600.koeffB)/StepsPe
r800;

InitialWave=Gratka600.koeffA;

}


if(Gratka==1200)

{

LPerStep=(800.0*Gratka1200.koeffB)/StepsP
er800;

InitialWave=Gratka1200.koeffA;

}

}
```

```
//-----------------------------------------------------
-------------------------

void      Foto::Monochromator::SetGratka(int
gr){//встановлює гратку 300 600 1200


Gratka=gr;

if(Gratka==300)

{

LPerStep=(800.0*Gratka300.koeffB)/StepsPe
r800;

InitialWave=Gratka300.koeffA;

}


if(Gratka==600)

{

LPerStep=(800.0*Gratka600.koeffB)/StepsPe
r800;

InitialWave=Gratka600.koeffA;

}


if(Gratka==1200)

{

LPerStep=(800.0*Gratka1200.koeffB)/StepsP
er800;

InitialWave=Gratka1200.koeffA;

}

}

//-----------------------------------------------------
-------------------------

void      Foto::Monochromator::ResetWave(int
ttime){


Form1->Memo1->Lines-
>Add(TimeToStr(Now())+": Встановлення 0
```

```
монохроматора " + IntToStr(StepMotorNumber));

//устанавл. длительность одного шага двигателя в мс.

if((MoveStep(/*-0xffff*/ -65535,ttime,RotateDirection))==-2)

{


MoveStep(InitialStep,ttime,RotateDirection); //назад на ~290 кроків установка "0" барабана


WaveLength= InitialWave;

StepCount=0;

Form1->Memo1->Lines->Add(TimeToStr(Now())+": Монохроматор " + IntToStr(StepMotorNumber) +" встановлено на 0");

}

else

{

Form1->Memo1->Lines->Add(TimeToStr(Now())+": Помилка при встановлені 0 монохроматора " + IntToStr(StepMotorNumber));

}

}

//---------------------------------------------------------

double Foto::Monochromator::CurrentWave(){

return  StepCount*LPerStep+InitialWave;

}

//---------------------------------------------------------

int Foto::Monochromator::CurrentStep(){// видає значення поточного кроку(від 0 поділки) двигуна

return StepCount;

}

//---------------------------------------------------------

double Foto::Monochromator::CurrentScaleDivision(){

return 800.0*(double)StepCount/(double)StepsPer800;

}
```

Додаток 5: Функції для відправки команд у Arduino та очікування повернення даних з мікроконтролеру та їхньої подальшої обробки

```
void SendCommand(int cmc,int StepAve,int StepTime)
{
	DWORD temp;

WriteFile(COMport,&cmc,2,&temp,&overlappedwr);

WriteFile(COMport,&StepAve,4,&temp,&overlappedwr);

WriteFile(COMport,&StepTime,2,&temp,&overlappedwr);
}
//-------------------------------------------

int COM_ReadCheck()
{
	while(TransferVar.flag==0){;};

	TransferVar.flag=0;

if((TransferVar.message[6]==76)||(TransferVar.message[6]==82)||(TransferVar.message[6]==77)||(TransferVar.message[6]==83))
		{
		Form1->Memo1->Lines->Add(TimeToStr(Now())+": Спрацював кінцевик!");
		//WaveLength=-1;
		TransferVar.flag=0;
		return -1;
		}

if((TransferVar.message[6]==79)&&(TransferVar.message[7]==75))
	{
	Form1->Memo1->Lines->Add(TimeToStr(Now())+": Крок виконанний успішно!"); /
	TransferVar.flag=0;
	return 1;
	}
	else
	{
	Form1->Memo1->Lines->Add(TimeToStr(Now())+": Помилка при виконанні кроку!");
	TransferVar.flag=0;
	return 0;
	}
```