Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Моделі імітаційного налагодження та верифікації мікроконтролерних систем логічного управління реального часу
(тема)

Виконав: студентка 2 курсу, групи СКСм-22-1

_____Шморгай К.О._____
(прізвище, ініціали)

Спеціальність *123 _Комп'ютерна інженерія*

Тип програми __*освітньо-професійна*_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи ___доц.Шкіль О.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_ Комп'ютерна інженерія _____
(шифр і назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва

ЗАТВЕРДЖУЮ:

Зав. кафедри ___  ___
(підпис)

«__02__»___09____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці_____Шморгай Катерині Олегівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Моделі імітаційного налагодження та верифікації мікроконтролерних систем логічного управління реального часу
Models of Simulation Debugging and Verification of Real-Time Logic Control Microcontroller Systems

затверджена наказом по університету від «__03_»__11__2023 р. № _1282 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____10.01.2023_____

3. Вихідні дані до роботи (проекту)_____

Протоколи передачі даних UART/USART, SPI, I^2C, CAN

Мікроконтролери ATMega, STM, ESP

Перептворювач інтерейсів FT2232

Мови програмування C, Python

Перелік питань, що потрібно опрацювати у роботі _____

Особливості налагодження мікроконтролерних систем у реальному часі

Імітаційне налагодження та тестування мікроконтролерних систем

Аналіз інтерфейсної частини мікроконтролерних систем та протоколів обміну з периферійними пристроями

Верифікація МК систем логічного управління у реальному часі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 19 слайдів

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 02.09.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2023-08.09.2023	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2023-15.09. 2023	
3	Аналіз технологій налагодження та діагностування систем на кристалі	16.09.2023-29.09.2023	
4	Вибір налагоджувальних плат та інтерфейсів зв'язку для реалізації проекту	30.09. 2023-20.10.2023	
5	Реалізація проекту керуючого автомату мовою програмування С	20.10. 2023-20.11. 2023	
6	Фізична реалізація проекту в налагоджуваих платах	20.11. 2023-10.12. 2023	
7	Проведення ДЕ	10.12. 2023-30.12. 2023	
8	Оформлення пояснювальної записки	01.01. 2024-11.01. 2024	
9	Захист проекту	12.01. 2024-25.01. 2024	

Студент _____
(підпис)

Керівник роботи (проекту) _____ доц.Шкіль О.С
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 73 сторінки, 31 рисунок, та 10 джерел за переліком посилань.

МІКРОКОНТРОЛЕР, ІНТЕРФЕЙС. НАЛАГОДЖУВАЛЬНА ПЛАТА, КІНЦЕВИЙ АВТОМАТ, ГРАФ ПЕРЕХОДІВ, МОВИ ПРОГРАМУВАННЯ С. PYTHON, ДІАГНОСТИЧНИЙ ЕКСПЕРИМЕНТ

В кваліфікаційній роботі розглянуті питання налагодження та діагностування мікроконтолерних систем логічного управління в реальному часі. При діагностиці систем логічного управління реального часу доцільно використовувати імітаційне налагодження при якому тестер імітує не тільки логічні умови алгоритму функціонування, а і оповіщувальни сигнали об'єкта управління в реальному часі. При цьому виділено три рівні імітації: інтерфейсний, процесорний і програмний рівень. Показано, що найбільш важливою частиною є інтерфейсний рівень налагодження.

В якості об'єктів діагностування розглянуті мікроконтролерні налагоджувальні плати Arduino, STM та ESP. В якості інтерфейсів зв'язку розглянуті протоколи UART, SPI, I^2C та CAN. В якості тестера запропоновано використовувати одноплатний комп'ютер, а для уніфікації протоколів та зв'язку з об'єктами діагностування використовувати інтегрований перетворювач інтерфейсів FT2232.

Діагностичний експеримент було проведено на моделі пристрою керування дорожним світлофором за допомогою персонального комп'ютера в якості тестера, налагоджувальних плат Arduino та STM, протоколів передачі даних UART та SPI і логічного аналізатора Salea Logic Analyzer.

ABSTRACT

The explanatory note contains: 73 pages, 31 figures, 10 sources according to the list of links.

MICROCONTROLLER, INTERFACE, DEBUGGING BOARD, FINITE STATE MACHINE, STATE DIAGRAM, C, PYTHON PROGRAMMING LANGUAGES, DIAGNOSTIC EXPERIMENT

In the qualification work, debugging and diagnosing issues of microcontroller logic control systems in real time were considered. During diagnosis of real-time logical control systems, it is advisable to use simulation debugging, when the tester simulates not only the logical conditions of the operating algorithm, but also the warning signals of the control object in real time. In this case, three levels of simulation are distinguished: interface, processor and software levels. It is shown that the most important part is the interface debugging level.

Microcontroller debugging boards Arduino, STM and ESP were considered as diagnostic objects. The UART, SPI, I^2C and CAN protocols are considered as communication interfaces. It is proposed to use a single-board computer as a tester, and to unify protocols and for communication with diagnostic objects – the integrated interface converter FT2232.

The diagnostic experiment was carried out on a model of a traffic light control device using a personal computer as a tester, debugging boards Arduino and STM, data transfer protocols UART and SPI and Salea Logic Analyzer.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЦП – аналогово-цифровий перетворювач;

ДЕ – діагностичний експеримент;

МК – мікроконтролер;

ПЗ – програмне забезпечення;

ПЗП – постійній запам’ятовуючий пристрій;

ПК – персональний комп’ютер;

ПЛІС – програмовані логічні інтегральні схеми;

САПР – система автоматизованого проектування;

СЛУ – системи логічного управління;

СРЧ – системи реального часу;

ШІМ – широтно-імпульсна модуляція;

ARM – Advanced RISC Machine (32-бітна процесорна архітектура);

BDM – Background Debug Mode (інтерфейс режиму фонового налагодження);

CRC – Cyclic redundancy check (метод обчислення контрольних сум);

FPGA – field-programmable gate array (різновид ПЛІС);

FSM – finite state machine (кінцевий автомат);

GPIO – General Purpose Input/Output (інтерфейс введення/виведення загального призначення);

IDE – Integrated Development Environment (інтегроване середовище розробки)

JTAG – Joint Test Action Group (спеціалізований апаратний інтерфейс на базі стандарту IEEE 1149.1);

RTL – register transfer level (рівень регістрових передач);

SoC – System on Chip (система на кристалі);

UUT – Unit Under Test (об’єкт діагностування).

ВСТУП

Складність застосування традиційних методів налагодження для систем логічного управління реального часу пов'язана з цілим рядом факторів. Найбільший вплив надає зростання складності сучасних проєктів, розміщених всередині одиночної інтегральної схеми, що програмується.

З розвитком комп'ютерної техніки імітаційне моделювання набирає все більшої популярності як для проведення досліджень, так і для побудови тренажерів. В даний час існує багато програмних пакетів, призначених для імітаційного моделювання в реальному часі. Використання таких пакетів для завдань налагодження апаратури та програм мікроконтролерних систем реального часу обмежено такими факторами:

- додаткові витрати часу вивчення середовища моделювання;
- невиправдана дорожнеча для об'єктів середньої складності;
- надмірна надмірність і складність стосовно поставлених завдань;
- наявність апаратних ресурсів (входів/виходів реального контролера) чи програмного інтерфейсу.

Перші три чинники створюють серйозний бар'єр для використання будь-якого програмного забезпечення, стандартного САПР або середовища імітаційного моделювання. Це пов'язано, в першу чергу, з різноманіттям складних технологічних систем, в ході реалізації яких можуть змінюватися параметри їх функціонування та структура технологічного циклу; складністю практичних завдань при оцінці рівня надійності та безпеки потенційно небезпечних промислових об'єктів; необхідністю врахування людського фактора при виконанні робіт на потенційно небезпечних об'єктах.

Виходячи з вищесказаного, для завдань налагодження найбільш практичним способом імітаційного моделювання є використання програмно-апаратних ресурсів контролера або одноплатного комп’ютера, в якому реалізовані алгоритми, що перевіряються.

1 ТЕХНОЛОГІЇ НАЛАГОДЖЕННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ ЛОГІЧНОГО УПРАВЛІННЯ РЕАЛЬНОГО ЧАСУ

1.1 Особливості налагодження мікроконтролерних систем логічного управління

Логічним управлінням називають управління технічними системами за допомогою дискретних сигналів, що приймають кінцеве число фіксованих значень, і здійснюване відповідно до заданого алгоритму управління. Зазвичай використовують дворівневі логічні сигнали, що позначаються умовно "0" та "1". Автоматизація логічного управління здійснюється за допомогою апаратних та програмних засобів, що отримали назву систем логічного управління (СЛУ). При побудові систем логічного управління як правило, виділяють пристрої, що керують, і керовані об'єкти. Керуючий пристрій реалізує алгоритм поведінки, тобто вибір виконуваних дій технічної системи, залежний від поточного стану і вхідних сигналів, а також перехід у новий технічний стан. В якості керуючого пристрою як правило, використовується часовий керуючий автомат (timed finite state machine, FSM). В якості технологічних платформ розробки СЛУ використовуються, як правило, мікроконтролерні (МК) системи. Особливо важливими є процедури налагодження мікроконтролерних систем управління в критичних технічних системах [1].

Мікроконтролери вирішують дуже різні завдання, в тому числі керування силовим електроустаткуванням, наприклад, перетворювачами частоти для електродвигунів, коректорами потужності, сервоприводами тощо. Обладнання, де протікають кілоампери і застосовуються кіловольти, де на рахунку кожна комутація ключів інвертора, де час реакції мікроконтролера на позаштатну ситуацію вимірюється в мікросекундах, а обладнання в герметичних корпусах встановлюється і експлуатується на

віддалених підприємствах має функціонувати сверхнадійно і потребує особливих методів налагодження [2].

По-перше, коли МК працює із силовим обладнанням, його не можна зупиняти. МК за допомогою ШІМ керує силовими ключами, регулює багато величин у своїх коридорах - струми фаз двигуна, напруга ланки постійного струму, частоту обертання, положення робочого органу тощо. Якщо зупинити МК під час роботи на точці зупинки, то в кращому випадку обладнання відключиться по апаратному захисту, а в гіршому все просто вибухне (якщо силові ключі залишаться включеними в одному положенні або з одним завданням шпаруватості). Тому класичний спосіб налагодження – «пройти кроками відладчиком» у таких завданнях не працює. Так можна лише обкатати «на столі» дуже сирі програмні модулі перед їх першим включенням на справжньому устаткуванні.

По-друге, це складність низькорівневого ПЗ. Розмір секції програмного коду .txt 50-200 Кбайт для програмного коду, що управляє виключно апаратною частиною (без високорівневих комунікаційних драйверів тощо) – це типова ситуація для мікроконтролера будь-якого промислового сервоприводу. Тому для таких завдань використовують мікроконтролери серії motorcontrol, які поєднують у собі одночасно і дуже розвинену периферію, і високу продуктивність, і відносно великий обсяг пам'яті. ПЗ для таких МК зазвичай містить десятки або сотні тисяч рядків добірного оптимізованого Сі/Сі++ коду, які просто неможливо налагодити «миготінням світлодіодом» або спостереженням роботи ніжок МК зовнішнім осцилографом.

По-третє, контролер системи керування електроприводом найчастіше стоїть усередині силового перетворювача, корпус якого закритий, а іноді навіть загерметизований. Тому доступ JTAG – вже велика проблема хоча б тільки для прошивки (з вимкненим силовим живлепнням). І ще серйозніша проблема - налагодження по JTAG з включеним силовим живлепнням. Тут повинен використовуватися обов'язково гальванічно-розв'язаний JTAG

(наприклад, для МК JTAG SAURIS з вбудованою гальванічною розв'язкою – дорого, але працює).

По-четверте, жодних операційних систем! Можна передбачити, як у деяких в голові крутиться думка "ну раз у вас таке складне завдання, поставте нормальний мікроконтролер з Linux і пишіть під нього звичайні програми, оновлюючи програмне забезпечення з флешки". Системи керування силовим обладнанням - системи дуже жорсткого реального часу. Не те, що Linux, навіть не всі спеціалізовані ОС реального часу підходять для таких завдань. Наприклад, переривання АЦП зчитування та усереднення аналогових даних може викликатися з частотою до 100кГц. При цьому воно міститиме всього десяток-другий рядків коду – збір даних аналогових каналів, пара перевірок швидкодіючих захистів і вихід - все більше нічого не встигнути. Якщо доручити таке переривання якомусь планувальнику завдань ОС, то його власне виконання буде займати більше ресурсів. Не кажучи вже про те, що, зокрема, для Linux замість однієї мікросхеми МК на плату треба поставити ще дві – зовнішню оперативну та флеш-пам'ять, відвівши на них купу дорогоцінних ніжок кристала, використовувати шестишарову плату для правильного розведення, отримати величезний час завантаження МК (іноді при збої живлення або спрацьовуванні сторожового таймера треба почати роботу раніше, ніж двигун встиг зупинитися!), мати проблеми з цілісністю файлової системи тощо.

Ну і п'ята особливість – системи низькорівневого управління силовим обладнанням (на відміну від завдань комунікації за різними інтерфейсами, контролерів будь-яких сигналізацій, пультів та іншого) реалізують в основному алгоритми систем автоматичного регулювання. А саме: софт наполовину складається з різноманітних замкнутих структур з ПІД-регуляторами, блоками насичення, мертвої зони, таблиць залежності одного від іншого, фільтрами, спостерігачами, задатчиками інтенсивності, планувальниками руху тощо. Обчислення в такому ПЗ виконуються з певною частотою – скажімо, на частоті 10кГц викликається переривання і обраховує

всі перелічені блоки, які утворюють разом потрібну структуру управління обладнанням. У таких алгоритмах покрокове налагодження не допомагає – найчастіше на кожному кроці кожен модуль видає те, що від нього чекають - власне, всі ці фільтри, регулятори та інше вже налагоджено роками, і сюрпризів там мало. Проблеми виникають під час роботи всієї структури загалом – кожен блок окремо працює, а бажаний процес регулювання відбувається негаразд, як очікується. І проблеми виявляються як у налаштуванні сотень параметрів і коефіцієнтів цих блоків, так і в самій зібраній з них структурі - може статися, що потрібно додати новий контур стабілізації десь, додати блок передбачення, обмеження тощо. Тому це чергове обмеження для покрокового налагодження та налагоджувальних повідомлень printf: налагоджувати потрібно найчастіше не сам програмний код, а зібрану структуру автоматичного управління. Особливо це важливо в системах реального часу.

1.2 Проектування та налагодження систем реального часу

Системою реального часу (СРЧ) є така система, коректність функціонування якої визначається як коректністю виконання обчислень, а й часом, коли отримано необхідний результат. Якщо вимоги за часом не виконуються, вважається, що відбулася відмова системи.

Для того щоб система могла задовольнити вимоги до систем реального часу, апаратні, програмні засоби та алгоритми роботи системи повинні гарантувати задані часові параметри реакції системи. Час реакції не обов'язково має бути дуже маленьким, але він має бути гарантованим (і таким, що відповідає поставленим вимогам). Майже всі системи промислової автоматизації є системами реального часу. Використання терміна «система реального часу», визначеного вище, для позначення інтерактивних і високопродуктивних систем неправильно [3].

Належність системи до класу систем реального часу ніяк не пов'язана з

її швидкодією. Наприклад, якщо ваша система призначена для контролю рівня ґрунтових вод, то навіть виконуючи вимірювання з періодичністю один раз за півгодини, вона працюватиме в реальному часі.

Термін «квазіреальний час» (soft real-time) хоча і використовується, але чітко не визначений. До його чіткого визначення навряд чи можливе його застосування у документах (крім рекламних). З упевненістю можна сказати, що сенс терміна «реальний час» трактується фахівцями по-різному в залежності від галузі їх професійних інтересів, від того, є вони теоретиками чи практиками, і навіть просто від особистого досвіду та спілкування;

Вихідні вимоги до часу реакції системи та іншим часовим параметрам визначаються або технічним завданням на систему, або логікою її функціонування. Наприклад, шахова програма, яка розраховує наступний хід більше години, явно працює не в реальному часі. Проте точне визначення «прийнятного часу реакції» який завжди є простим завданням, а системах, де однією з ланок служить людина, піддається впливу суб'єктивних чинників. Інтуїтивно зрозуміло, що швидкодія системи реального часу має бути тим більшою, чим більша швидкість перебігу процесів на об'єкті контролю та управління.

Отже, системою реального часу називатимемо апаратно-програмний комплекс, що реагує за передбачуваний час на непередбачуваний потік зовнішніх подій [4].

Це означає, що СРЧ має встигнути відреагувати на подію, що сталася на об'єкті, протягом часу критичного для цієї події. Величина критичного часу для кожної події визначається об'єктом і самою подією і, природно, може бути різною, але час реакції системи має бути передбачено (обчислено) під час створення системи. Відсутність реакції у передбачений час вважається помилкою для систем реального часу. СРЧ має встигати реагувати на події, що відбуваються одночасно. Навіть якщо дві чи більше зовнішні події відбуваються одночасно, система повинна встигнути зреагувати на кожну з них протягом інтервалу часу, критичного для цих

подій.

Розрізняють системи реального часу двох типів – системи жорсткого реального часу та системи м'якого реального часу.

1. Системи жорсткого реального часу не допускають жодних затримок реакції за жодних умов, оскільки:

- результати можуть виявитися неактуальними у разі запізнення;
- може статися катастрофа у разі затримки реакції;
- вартість запізнення може бути нескінченно велика.

Приклади систем жорсткого реального часу – бортові системи керування, аварійного захисту, реєстратори аварійних подій.

Багато теоретиків ставлять тут точку, з чого випливає, що час реакції в «жорстких» системах може становити і секунди, години і тижні. Однак більшість практиків вважають, що час реакції в системах «жорсткого» реального часу має бути мінімальним. Зрозуміло, однозначної думки у тому, який час реакції властиво «жорстким» системам, немає. Більше того, з збільшенням швидкодії мікропроцесорів – цей час має тенденцію до зменшення, і якщо раніше як межа називалося значення 1 мс, то зараз, як правило, називається час порядку 100 мкс.

2. Системи м'якого реального часу характеризуються тим, що затримка реакції не критична, хоча може призвести до збільшення вартості результатів і зниження продуктивності системи загалом. Приклад – робота комп'ютерної мережі. Якщо система не встигла обробити черговий прийнятий пакет, це призведе до тайм-ауту на стороні, що передає, і повторній посилці пакету (залежно від протоколу, звичайно). Дані при цьому не втрачаються, але продуктивність мережі знижується.

Основну відмінність між системами жорсткого та м'якого реального часу можна виразити так: система жорсткого реального часу ніколи не запізниться з реакцією на подію, а система м'якого реального часу – не має запізнюватися з реакцією на подію.

Методики проектування та налагодження систем реального часу

використовують методологію створення мікропроцесорних та мікроконтролерних систем та мають певну специфіку. Мікроконтролерні системи орієнтовані на виконання завдань управління певними пристроями чи його комплексами. Мікропроцесорні системи можна умовно розділити на два основні класи: універсальні, які використовуються для вирішення широкого кола завдань обробки інформації, та керуючі, що спеціалізуються на вирішенні завдань управління процесами та об'єктами. Типовими прикладами універсальних мікропроцесорних систем є персональні комп'ютери та робочі станції, які застосовуються у різних сферах діяльності.

Керуючі мікропроцесорні системи мають багато спільного з мікроконтролерними. Вони зазвичай використовуються для реалізації складних алгоритмів управління, що вимагають великої обчислювальної потужності процесора, яку не можуть забезпечити мікроконтролери. При цьому периферійні пристрої, багато з яких розташовуються на кристалі мікроконтролера, мікропроцесорних системах реалізуються за допомогою додаткових мікросхем, що підвищує їх вартість і знижує надійність. Розробка інтегрованих мікропроцесорів, що мають у своєму складі ряд периферійних пристроїв, і складно-функціональних мікроконтролерів, що містять високопродуктивне 32-розрядне процесорне ядро, призводить до розмивання межі застосування мікропроцесорних і мікроконтролерних систем, що управляють, поступового стирання функціональних і структурних відмінностей між ними [5].

Основною особливістю мікроконтролерів є наявність у їхньому складі ПЗП, до якого записується резидентна робоча програма системи. Розробка, налагодження та запис у ПЗП цієї програми є найважливішими стадіями проектування мікроконтролерних систем. Записана в ПЗП робоча програма стає складовою системи, наступна зміна чи корекція якої зазвичай небажана чи неможлива. З використанням внутрішнього ПЗП можливості зовнішнього контролю роботи мікроконтролера у процесі налагодження дуже обмежені. Тому комплексне налагодження програмного та апаратного забезпечення

мікроконтролерних систем є досить складною процедурою, що вимагає використання спеціалізованих методів та засобів контролю. Даний етап проектування є найбільш відповідальним, тому що не виявлена помилка може призвести до дуже дорогих наслідків. Особливістю мікроконтролерів для ряду областей застосування є необхідність суворого дотримання певних норм часу на виконання програми або її окремих модулів.

Етап розробки апаратури може бути виконаний традиційними методами, за допомогою яких проектується та моделюється електрична схема, розробляється друкована плата або комплект плат, після чого виконується монтаж та налагодження системи. Однак у багатьох випадках можна забезпечити скорочення термінів та підвищення якості розробки апаратури шляхом використання "напівфабрикатів" або готових виробів, що випускаються рядом виробників.

Існує досить велика номенклатура таких виробів, які носять, назви оціночних чи цільових плат (evaluation board, target board), оціночних наборів чи систем (evaluation kit, evaluation system), одноплатних комп'ютерів чи контролерів (SBC-single-board computer, single-board controller). Ці вироби, як правило, називають платами налагодження. До їх складу входить базовий мікропроцесор або мікроконтролер, пам'ять (ОЗУ, флеш-пам'ять, службове ПЗП), ряд периферійних та допоміжних схем. Зазвичай такі плати мають з'єднувач для підключення до персонального комп'ютера, за допомогою якого здійснюється комплексне налагодження системи.

Якщо склад засобів, що є на платі налагодження, достатній для реалізації системи, що проектується, то її розробка зводиться до створення та виконання комплексного налагодження системи. Якщо наявних засобів недостатньо, вони проектуються і розміщуються на додатковій платі, що підключається до з'єднувачу на платі налагодження безпосередньо чи за допомогою кабелю. Так реалізується прототип проектованої системи, на якому можна виконати комплексне налагодження програмних та апаратних засобів, а в ряді випадків і провести перевірку їхнього функціонування в

робочих умовах. Після цього розробляється робочий варіант системи, що об'єднує на одній платі модулі прототипної системи, що використовуються. Прототипна система може використовуватися як робоча, якщо її параметри та конструктивне оформлення задовольняють вимогам технічного завдання. У цьому випадку досягається скорочення термінів та вартості проектування та налагодження системи.

На етапі автономного налагодження апаратури основними знаряддями розробника є традиційні вимірювальні прилади – осцилографи, мультиметри, пробники та інші, а також логічні аналізатори, які мають широкі можливості контролю стану різних вузлів системи в задані моменти часу. Дуже ефективним є використання на цьому етапі засобів тестування за стандартом JTAG, які є у складі багатьох сучасних моделей мікропроцесорів та мікроконтролерів. За допомогою розміщеного на кристалі тест-порту TAP та спеціальних висновків TDI, TDO, TCK, TMS, TRST# забезпечується можливість подачі необхідних вхідних впливів та зчитування вихідної реакції, запуск-зупинок процесора, зміна режиму його роботи. Введенням спеціальної команди можна встановити виводи мікропроцесора або мікроконтролера у вимкнений стан, щоб окремо протестувати інші пристрої системи.

Автономне налагодження програмного забезпечення виконується за допомогою симулятора – програмної моделі використовуваного мікропроцесора або мікроконтролера. На цьому етапі розробники використовують широкий набір засобів програмування - компілятори, асемблери, дисассемблери, відладники, редактори зв'язків та інші, без яких практично неможливо створення працездатного програмного забезпечення протягом обмежених термінів виконання проекту.

Комплексне налагодження апаратури та ПЗ є найбільш складним та відповідальним етапом створення системи. На цьому етапі розробник використовує весь набір програмних та апаратних засобів, що застосовуються для автономного налагодження, а також ряд спеціальних

засобів комплексного налагодження. До таких засобів відносяться схемні емулятори – спеціалізовані пристрої, що включаються замість мікроконтролера прототипної системи і забезпечують можливість контролю її роботи за допомогою персонального комп'ютера, пов'язаного зі схемним емулятором. Схемні емулятори є найефективнішим засобом комплексного налагодження систем.

Слід зазначити, що багато моделей мікропроцесорів та мікроконтролерів мають спеціальний режим фонового налагодження (Background Debug Mode, BDM) при якому реалізується введення команд, введення-виведення даних, управління режимом роботи процесора за допомогою послідовного спеціального порту. При його використанні мікропроцесор або мікроконтролер може працювати в режимі емуляції під керуванням персонального комп'ютера, що підключається до цього порту. Режим BDM дозволяє суттєво полегшити процедуру комплексного налагодження та використовувати при цьому простіші та дешевші засоби.

Одним із найбільш ефективних засобів комплексного налагодження мікроконтролерних систем є емулятори ПЗП. Ці пристрої включаються замість ПЗП прототипної системи та працюють під керуванням підключеного до них персонального комп'ютера. Так забезпечується поточний контроль за виконанням програми та її оперативна корекція, що значно спрощує процес налагодження.

Для мікроконтролерних систем заключною процедурою комплексного налагодження є запис у ПЗП об'єктних модулів налагодженої програми та завершальне випробування її працездатності. Запис програми у ПЗУ здійснюється за допомогою спеціальних програматорів.

Після виконання зазначених етапів налагоджена система, що прототипується, може бути випробувана в робочих умовах із підключенням повного набору реальних периферійних пристроїв та об'єктів керування. У процесі дослідної експлуатації виявляються помилки, не виявлені на етапі налагодження, визначається реакція системи на можливі непередбачені

ситуації. У процесі розробки при створенні сучасних мікропроцесорних та мікроконтролерних систем використовується комплекс програмно-апаратних засобів, які допомагають якісно та в обмежені терміни виконати їх проектування та налагодження.

1.3 Імітаційне налагодження та тестування мікроконтролерних систем

Проектування програмно-апаратних комплексів, що складаються з технічно складних апаратних пристроїв, а також реалізують набір програм, що виконуються на одному або декількох мікроконтролерних пристроях, є складним завданням. Проблема контролю та діагностики несправностей має актуальність для таких мікроконтролерів, як критичні системи управління, інформаційно-обчислювальні кластери, оскільки для цих систем існують жорсткі вимоги до такого показника надійності, як час напрацювання на відмову. Вирішення проблеми контролю та діагностики програмно-апаратних засобів ускладнюється тим, що ці системи схильні до різноманітних зовнішніх впливів, повний контроль яких фізично нереалізований. Слід зазначити, що окремою проблемою є проблема стійкості до відмови програмного забезпечення, яке, на жаль, не можна розглядати незалежно від апаратних засобів, на яких воно реалізовано. У процесі випробувань вироби на надійність не завжди вдається коректно встановити всі несправності у програмній та апаратних складових, тому при експлуатації комплексу пристроїв можливі економічні втрати та невиконання встановленого у технічному завданні показника часу безвідмовної роботи [6].

Для ефективної діагностики відмов та збоїв мікроконтролерної програмно-апаратної системи потрібно виділити основні типи причин, що викликають появу відмови чи збою та розробити комплекс показників оцінки відмов та збоїв. Визначимо основні типи несправностей:

- апаратний збій – тимчасовий вихід із ладу компонента мікроконтролера з подальшим поверненням до нормального режиму роботи;

- апаратна відмова – вихід із ладу компонента мікроконтролера та неможливість його подальшої роботи без заміни компонентів або резервування;
- відсутність робастності – характеризує нестійкість системи до вхідних даних, з невизначеним технічним завданням (під робастністю розуміється властивість стійкості системи за непередбачених вхідних даних);
- порушення часових параметрів в інтерфейсах зв'язку периферійних пристроїв із центральним мікроконтролером;
- помилки програмного забезпечення (ПЗ) – ненавмисне відхилення програми від заданого виконання внаслідок внутрішніх та зовнішніх факторів (помилки компіляторів, програмістів, вплив апаратних відмов та збоїв тощо).

Виявлення апаратних відмов вимагає використання апаратних модулів керування, які визначають технічні несправності компонентів. Одним із ефективних підходів є порівняння сигналів на входах і виходах і параметричне регулювання модуля. При відхиленні параметрів модуля від встановленого порядку відбувається фіксація несправності, після чого інформація передається на робочу станцію. Застосування таких модулів актуально для основних підсистем мікроконтролера: систем керування внутрішньою пам'яттю, шин даних, блоків введення-виведення, апаратного таймера. Модулі працюють по тестових програмах, які змінюють значення параметрів, якими керує модуль.

Імітаційне налагодження та діагностування мікроконтролерних систем пов'язано з вибором необхідних компонентів серед усіх методів і засобів, що застосовуються до мікроконтролерних пристроїв. Тому комплекс заходів у таких системах складається з іншого набору інструментів, які можна виділити в спеціалізовану архітектуру. Виділяється основні рівні імітації при діагностуванні: інтерфейсний, процесорний і програмний рівень.

Методи верифікації мікроконтролерів можна розділити на три основні класи: експертизу, формальну верифікацію та імітаційне тестування. Крім

цих методів існують звані синтетичні чи гібридні методи, використовують комбінації різних підходів [7].

До експертизи належать методи верифікації, у яких оцінка результатів проектування виконується людьми шляхом безпосереднього аналізу. Від інших методів експертизу відрізняє можливість виконувати її, використовуючи тільки результати проектування, а не їх формальні моделі (як у формальній верифікації) або результати роботи (як в імітаційному тестуванні). Формальні методи верифікації використовують формальні моделі мікропроцесорів та його модулів. Це можуть бути логічні або алгебраїчні моделі, моделі на основі графів чи кінцевих автоматів, а також моделі інших типів. Формальні методи здатні виявляти складні помилки, що практично не виявляються за допомогою експертиз або тестування.

Імітаційним тестуванням називається тестування моделей, одержаних на різних етапах проектування. Для стислості будемо називати імітаційне тестування просто тестуванням, хоча слід розуміти, що під тестуванням зазвичай розуміють перевірку мікросхем щодо наявності технологічних помилок (обривів провідників, замикань тощо), що, безумовно, верифікацією не є. Для застосування тестування необхідно мати модель мікроконтролера або хоча б деяких модулів, тому тестування не можна використовувати на ранніх фазах проектування. Для проведення тестування потрібна додаткова підготовка: створення тестів та розробка тестової системи, що дозволяє їх виконувати.

Тестова система це спеціалізована програма, написана тією ж мовою, що й компонент, що тестується (Verilog, VHDL), мовою програмування загального призначення (C, C++), або спеціалізованою мовою верифікації апаратури (SystemC, SystemVerilog). Тестова система подає на компонент, що тестується, тестові сигнали (встановлюючи вхідні сигнали або, якщо тестовою системою є програма, створюючи за допомогою команд певні ситуації в роботі мікроконтролера) та здійснює перевірку правильності реакцій на них (зчитуючи вихідні сигнали або перевіряючи значення

регістрів). У процесі тестування тестовою системою створюється тестовий звіт (траса тесту), який використовується для аналізу помилок та оцінки повноти тестування. У загальному випадку тестова система вирішує три основні завдання: генерує тестову послідовність, перевіряє правильність поведінки компонента, що тестується та оцінює повноту тестування.

В даний час існує багато методів тестування, які по-різному підходять до генерації тестової послідовності, перевірки правильності поведінки та оцінки повноти тестування. Кінцеві автомати є широко використовуємим формалізмом для представлення апаратних систем і часто використовуються для генерації тестової послідовності. Як правило, тести будуються шляхом обходу графа станів автоматної моделі компонента, що тестується.

Навіть якщо тестові послідовності охоплюють всі ситуації в роботі компонента, що тестується, помилки неможливо виявити, не проаналізувавши правильність поведінки компонента. Існує три основні методи перевірки правильності поведінки: ко-симуляція, самоперевіряючі тести та формальні специфікації. Для ко-симуляції окремо від тестованого компонента розробляють його модель, яка має ту ж функціональність, але представлена в більш абстрактній формі (еталонна модель). Якщо знаходиться невідповідність між поведінкою компонента, що тестується, і еталонної моделі, з'ясовують, яка модель некоректна. Після виправлення помилки тестування продовжується. Коли тести потрібно отримати швидко, замість ко-симуляції застосовують підхід з використанням тестів, що самоперевіряють. Даний спосіб передбачає, що кожен тест, крім тестової послідовності, містить перевірки, які необхідно провести після або під час на компонент. З одного боку, при використанні самоперевіряючих тестів не потрібна еталонна модель, але, з іншого боку, потрібні великі зусилля на розробку тестів, оскільки їх необхідно забезпечити відповідними перевірками.

Оцінюючи повноту тестування очевидно, що вичерпний перебір значень вхідних сигналів при всіляких станах мікроконтролера неможливий

і, взагалі кажучи, не потрібен (таке тестування є надлишковим). На практиці потрібен метод оцінки повноти тестування – критерій, який на основі значень деяких метрик дозволяє визначити, коли можна завершити тестування та передати МК замовнику. Часто використовується підхід, коли метрики повноти тестування визначають з урахуванням коду RTL-моделі мікропроцесора чи формальної специфікації – такі метрики називаються структурними і функціональними, відповідно. Критерієм повноти тестування зазвичай є досягнення 100% покриття ситуацій в рамках обраної метрики. На практиці для оцінки повноти тестування використовують комбіновані підходи, що враховують технічні, статистичні та економічні показники.

В роботі [8] запропоновано в якості тестера при налаштуванні проектів програмно-апаратних комплексів використовувати зовнішній комп'ютер, поєднаний з платою налаштування через інтерфейс JTAG (рис. 1.1). Найбільш повну картину поведінки зовнішніх і внутрішніх сигналів системи, що налагоджується, можна отримати, розмістивши всередині її логічний аналізатор, підключений до схеми, що проектується. Таке завдання легко вирішується при налагодженні проектів, які розміщені всередині налагоджувальної плати МК, в якій завжди залишаються невикористані ресурси.

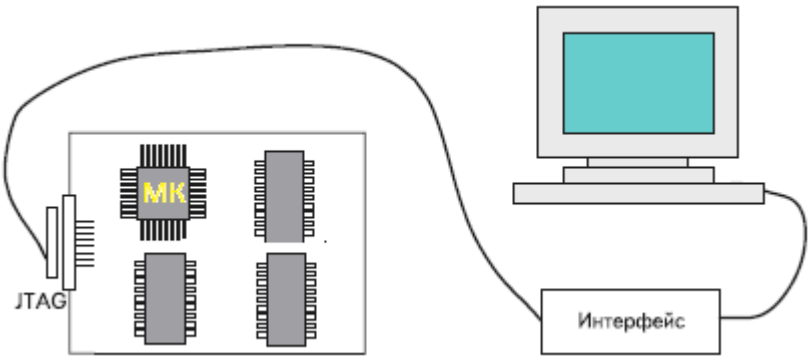

Рисунок 1.1 – Схема налагодження через JTAG

Але у багатьох алгоритмах логічного управління промисловими системами покрокове налагодження не дуже допомагає – найчастіше на

кожному кроці кожен модуль видає те, що від нього чекають - власне, всі ці фільтри, регулятори та інше вже налагоджено роками, і сюрпризів там мало. Проблеми виникають під час роботи всієї структури загалом – кожен блок окремо працює, а бажаний процес регулювання відбувається не так як очікується. Таким чином налагодити структуру логічного управління можна лише переглядом осцилограм та часових діаграм внутрішніх змінних, тобто. переглядом графіків у часі, як змінюється та чи інша змінна програми МК.

Крім того інтерфейс JTAG має два суттєвих недоліки. По-перше, JTAG із-за низької швидкості не дозволяє отримати осцилограми або діаграми логічного аналізатора в оперативному режимі на реальній частоті роботи пристрою, що налагоджується. По-друге, застосування JTAG обумовлює знаходження оператора-налагоджувальника безпосередньо на місці реального функціонування промислової мікроконтролерної системи, але там можуть бути не дуже комфортні умови.

Для вирішення цих проблем запропоновано часові діаграми логічного аналізатора записувати в оперативну пам'ять МК просто в масив. Зазвичай багато точок не треба - досить зловити потрібний момент, коли записувати дані: звести потрібний тригер на старт запису. І тоді можна побачити, що видавали ті чи інші блоки системи управління у момент збою, як він розвивався, що намагався зробити МК.

В результаті, все що робив оператор при налагоджуванні через JTAG замінюється на будь-який інший інтерфейс зв'язку і створюється своя оболонка верхнього рівня, яка відновлює потрібні часові діаграми безпосередньо на робочому місці оператора. Одним з шляхів вирішення є використання інтерфейсу зв’язку CAN (Controller Area Network). Це дуже хороший промисловий інтерфейс зв'язку, завадостійкий, має апаратний арбітраж, що не руйнує доступ до шини, апаратне підтвердження посилок, і при цьому – всього два дроти та земля. Основна функція CAN, як і багатьох протоколів для МК, це надання доступу до словника об'єктів (списку змінних на Сі МК) за певною адресою у вигляді постійного відправлення значень

таймером або подією. Також є різні службові послуги типу аварійного повідомлення (emergency), повідомлення наявності пристроїв у мережі (heartbeat) тощо [2].

Таким чином, вийшло теж саме, що й при налагодженні через JTAG, але віддалено та з додатковою функцією коригування прошивки МК. Тому Для цього був реалізований свій «програматор» через мережу CAN, але в рамках протоколу CANOpen (рис. 1.2).

Рисунок 1.2 – Схема налагодження МК через інтерфейс CAN

Звичайно, кожна серйозна фірма-розробник має свій власний інструментарій для подібної налагодження. Для потужних мікроконролерів широко використовується Ethernet, надаючи доступ до пристрою TCP/IP. Якщо мікроконтролер ще потужніший і містить зовнішню флеш-пам'ять, можна використовувати операційну систему, скажімо з урахуванням Linux, і тоді можливості з налагодження виходять принципово новий рівень. Деякі для налагодження використовують дисплей, підключений безпосередньо до мікроконтролера.

1.4 Постановка мети та завдань дослідження

Зі збільшенням областей застосування мікроконтролерних систем зростає ймовірність та тяжкість помилок, що виникають при програмуванні мікроконтролерів. Тому важливу роль при проектуванні

мікрококонтролерних систем логічного управляння відіграють налагоджувальні плати.

Налагоджувальні плати є друкованими платами зі вбудованим програматором, роз’ємами для одного або декількох мікроконтролерів і базовим набором периферії. Зазвичай включають світлодіоди, цифрові індикатори та перемикачі. Як правило такі плати виробляють самі виробники мікроконтролерів. На плати також встановлюють схеми зв'язку з комп'ютером, розведення для підключення плат розширення, макетну область для монтажу прикладних схем користувача.

При випуску нової налагоджувальної плати фірма-виробник випускає і відповідні плати розширення, наприклад, ПЗП, дисплей, клавіатуру, датчики, перетворювачі інтерфейсів, приймачі, відеокамери тощо. Крім навчальних цілей плати широко застосовуються в малосерійному виробництві як вбудовані плати управління. Ціна налагоджувальної плати залежить від функціональних можливостей.

Для повноцінної розробки проектів мікроконтролерних систем передбачається наявність трьох компонентів: налагоджувального середовища, емулятора (тестера) та налагоджувальної плати. Налагоджувальне середовище – це програмне забезпечення, створене для написання та налагодження програм під мікроконтролери. За допомогою емулятора до персонального комп'ютера (ПК) через відповідний інтерфейс підключається і програмується фізична налагоджувальна плата.

В спрощеному варіанті в якості тестера замість класичного ПК може використовуватися одноплатний комп’ютер (типу Raspberry Pi), який через відповідній інтерфейс зв’язку взаємодіє з мікроконтролерною налагоджувальною платою, що тестується, що зображено на рис. 1.3.

Рисунок 1.3 – Схема апаратного тестування МК на налагоджувальній платі

При такій схемі підключення однакових сімейств, серій, лінійок мікроконтролерів досить просте. Як правило узгоджується використанням одного й того ж мікроконтролера на різних платах розробки. Це дозволяє зробити підключення будь-яким типом без додаткової програмної чи апаратної реалізації. Достатньо підключити Тестер і МК дротово за принципом «pin-to-pin». Це може бути і протокол передачі даних, який має окремі піни на платах. Або ж просто підключення і використання периферії плат як GPIO і передача даних зводиться до подачі низького «0» або високого «1» рівня сигналу від Тестера до МК

Якщо ж необхідно підключити різні мікроконтролери будь-якого спектру (різні сімейства, серії або ж лінійки), варто ще до розробки рішення виокремити характеристики обраних інтерфейсів зв’язку тестера та МК пристроїв управління. Такими характеристиками являються.

1. Узгодження рівнів напруги Тестера та налагоджувальної плати. Якщо не дотримуватися даного правила, то існує висока ймовірність виведення з ладу будь-якої ланки рішення або плати розробки у цілому.

2. Частота мікроконтролерів. Необхідно зазначити, що ПУ з різними частотами процесора можуть значно погіршити або порушити коректність роботи рішень, в яких швидкість обробки даних або ж роботи самої програмної реалізації є найважливішою характеристикою. Звичайно, необхідно заздалегідь узгодити яким має бути рішення. Проте для тестування рішень, де швидкість обробки даних є другорядною або не найважливішою

складовою, допустимо використати плату, що тестує з мікроконтролером, частота якого може бути менша за частоту плати, що тестується. Наприклад, стандартна частота плати Arduino є 16МГц. Стандартна частота плати на базі STM32F103 – 72МГц. На рівні інтерфейсу це має бути узгоджено

3. Узгодження периферії. Якщо в рішенні існує специфічна периферія (наприклад певний протокол передачі даних (CAN, LIN тощо) чи робота з контролюванням напруги на периферії (ШІМ)), необхідно, щоб Тестер також мав її апаратну та програмну реалізацію. Інакше коректність роботи даного рішення близька до нуля. Вагомою перевагою також можна назвати кількість GPIO на платах. Це дозволяє узгодити специфічну периферію, так як найчастіше з платами, де кількість GPIO однакова, йде вся необхідна апаратна реалізація.

4. Спосіб передачі даних. Останній з найважливіших пунктів узгодження для ефективної та коректної роботи рішень. Дані для тестування можуть відправлятися за допомогою протоколу передачі даних (UART/USART, SPI, I^2C, CAN тощо). Протокол передачі даних може бути синхронним або асинхронним, послідовним або паралельним. Як і з третім пунктом узгодження дана периферія має бути реалізована на Тестері та мати той самий логічний рівень передачі даних, що і на налагоджувальній платі.

Всі пункти, опис яких знаходиться вище, реалізуються шляхом використання додаткової апаратної реалізації, у вигляді окремо реалізованих модулів-інтерфейсів передачі даних, перетворювачів рівнів напруги, розширювачів портів, додаткової пам’яті тощо. Це розповсюджена практика, яка дозволяє покрити всі важливі та другорядні узгодження. Проте варто зазначити, чим більша кількість апаратної реалізації, тим більша кількість апаратних та програмних витрат.

Метою роботи є розробка моделей аналізу та вибору інтерфейсів в системах імітаційної верифікації мікроконтролерних систем логічного управління реального часу.

Для досягнення поставленої мети необхідно вирішити такі задачі :

− визначити особливості тестової верифікації мікроконтролерних систем реального часу;

− проаналізувати існуючі інтерфейси мікроконтролерних систем та розробити критерії їх застосування для різних апаратних платформ;

− розробити процедури імітаційного моделювання автоматних систем логічного управління в реальному часі;

− розробити вимоги до тестера в мікроконтролерній системі імітаційного моделювання та універсального інтерфейсного блоку;

− провести діагностичний експеримент (ДЕ) з різними протоколами зв’язку в мікроконтролерних системах імітаційного моделювання.

2 АПАРАТНА ОРГАНІЗАЦІЯ ТЕСТУВАННЯ МК НА НАЛАГОДЖУВАЛЬНІЙ ПЛАТІ

2.1 Налагоджувальні плати МК

2.1.1 Сімейство плат Arduino

Arduino – це популярна лінійка відкритих апаратних плат, яка використовується для створення прототипів електронних проектів. Сімейство плат Arduino складається з різних моделей, призначених для різних завдань і рівнів складності.

Однією з найперших та найбільш популярних моделей є Arduino Uno. Ця плата має вбудований мікроконтролер Atmega328P, а також цифрові та аналогові входи/виходи, які дозволяють зручно підключати різні сенсори та пристрої. Arduino Uno використовується як основна плата для багатьох початкових проектів та входження в світ електроніки.

Іншими популярними моделями є Arduino Mega, яка має більше входів/виходів і призначена для більш складних завдань, та Arduino Nano, яка є компактною версією для проектів з обмеженим простором.

Arduino Due відзначається більш потужним мікроконтролером та можливістю працювати на вищих частотах, що дозволяє реалізувати більш складні програми та обчислення.

Крім того, існує сімейство Arduino плат, орієнтованих на специфічні галузі, такі як Arduino Nano Every для навчання програмуванню або Arduino MKR WiFi для проектів, які вимагають бездротового підключення до мережі. Arduino Lilypad призначена для проектів у сфері електроніки одягу та портативних пристроїв, завдяки своєму компактному розміру та можливості легкої інтеграції в текстиль. Arduino Yun поєднує в собі можливості звичайної Arduino з Wi-Fi з'єднанням та процесором Linux, що дозволяє

створювати інтернет-речей (IoT) проекти та обробляти дані в хмарі. Крім того, існують варіації Arduino з вбудованими сенсорами, такими як гіроскопи, акселерометри та датчики температури, що розширюють можливості для розробки проектів у галузі робототехніки, вимірювань та інших областей.

Arduino надає величезну кількість бібліотек та ресурсів для спрощення розробки, що робить її ідеальним вибором для ентузіастів, студентів та професіоналів в галузі електроніки. Завдяки своїй простоті та доступності, Arduino створює місто для креативності, дозволяючи реалізовувати ідеї та інновації в світі технологій.

Arduino використовує відкритий код і активної спільноти користувачів, яка обмінюється досвідом, програмним забезпеченням та ідеями. Це сприяє поширенню та підтримці платформи, а також розвитку нових можливостей для творчості.

Усі ці аспекти роблять Arduino не лише потужним інструментом для проектування електроніки, але й сприяють формуванню спільноти та культури, що стимулює інновації та навчання. Сімейство плат Arduino залишається важливою та захопливою частиною сучасного світу електроніки.

Розглянемо детальніше одну з найкомпактніших плат сімейства Arduino. Arduino Nano – це компактна та потужна плата мікроконтролера, яка входить в сімейство Arduino. Ця модель була створена з метою забезпечення користувачам можливості розробки проектів у обмежених просторах, а також для застосувань, де важливі розмір та невелика вага пристрою.

Основні характеристики Arduino Nano наступні.

Мікроконтролер: Arduino Nano використовує мікроконтролер ATMega328, той самий, що і в Arduino Uno. Це забезпечує сумісність з багатьма програмами та бібліотеками, розробленими для Arduino Uno.

Вхід/вихід: Плата має 14 цифрових входів/виходів, 8 аналогових входів та можливість розширення цифрових входів/виходів за допомогою додаткових модулів.

Роз'єми: Arduino Nano оснащена роз'ємами для підключення USB-кабелю, який використовується для програмування та живлення плати. Також є роз'єми для живлення та передачі даних через UART.

Розмір і форм-фактор: Однією з важливих особливостей Arduino Nano є її компактний розмір. Вона має форм-фактор, аналогічний багатьом іншим моделям Arduino, проте її розмір значно менший, що робить її ідеальним вибором для вбудованих систем та пристроїв з обмеженим простором.

Напруга живлення: Зазвичай Arduino Nano працює в діапазоні напруги від 5 до 12 вольт. Вона може бути живлена як від зовнішнього джерела, так і від USB-порту комп'ютера.

Сумісність та програмування: Arduino Nano програмується за допомогою Arduino IDE, що робить його доступним для широкого кола користувачів. Його сумісність з Arduino Uno дозволяє використовувати існуючі коди та бібліотеки.

Arduino Nano знайшла широке застосування в різних проектах, таких як розробка вбудованих систем, портативних пристроїв, робототехніки, сенсорних пристроїв та багато інших сфер. Її популярність полягає в зручності використання та широких можливостях для реалізації творчих ідей в галузі електроніки (рис. 2.1).

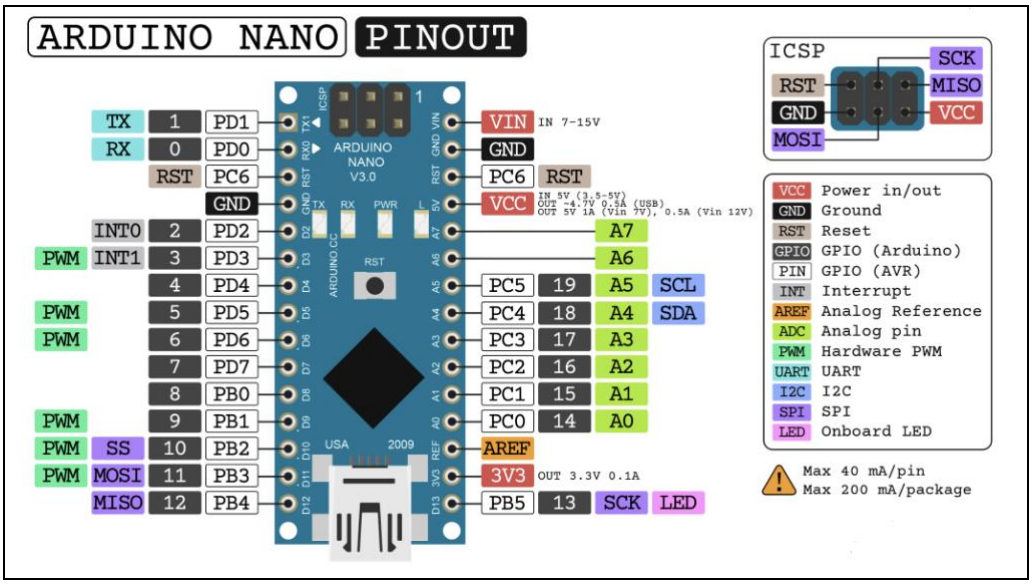

Рисунок 2.1 – Схематичне зображення плати розробки Arduino Nano

2.1.2 Сімейство плат STM32

Сімейство мікроконтролерних плат STM (STMicroelectronics) представляє собою широкий спектр високоефективних та потужних пристроїв, які використовуються для розробки вбудованих систем та інших електронних пристроїв. Ці плати відомі своєю надійністю, високою продуктивністю та широким набором можливостей, що дозволяє їх використовувати у різних застосуваннях.

STM32 є серією мікроконтролерів від компанії STMicroelectronics, і вона включає в себе широкий асортимент моделей із різними технічними характеристиками для задоволення різноманітних потреб розробників.

Основні характеристики сімейства мікроконтролерів STM32 наступні.

Архітектура ARM Cortex-M. Більшість мікроконтролерів STM32 базуються на ядрах ARM Cortex-M, таких як Cortex-M0, Cortex-M3, Cortex-M4 або Cortex-M7. Це забезпечує високу продуктивність та відмінні характеристики вартості енергії.

Широкий спектр моделей та серій. Сімейство включає в себе різні серії, такі як STM32F0, STM32F1, STM32F2, STM32F3, STM32F4, STM32F7, STM32L0, STM32L1, STM32L4, STM32H7 та інші. Кожна серія призначена для різних застосувань та відзначається своїми характеристиками.

Пам'ять. Мікроконтролери STM32 мають величезну різноманітність обсягів вбудованої флеш-пам'яті для зберігання програмного коду та вбудованих даних. Деякі моделі також мають вбудовану оперативну пам'ять.

Периферійні пристрої. STM32 оснащені різноманітними периферійними пристроями, включаючи порти GPIO, таймери, UART, SPI, I2C, CAN, USB, Ethernet, ADC, DAC, інші.

Живлення. Мікроконтролери можуть працювати при різних рівнях напруги живлення, що робить їх гнучкими в різних умовах застосування.

Розширення можливостей. Деякі моделі STM32 мають можливість підключення розширювальних модулів, таких як шилди або інші

розширювальні плати, що полегшують розширення функціональності.

Інструменти розробника. STMicroelectronics надає набір інтегрованих інструментів для розробки, таких як STM32CubeMX для конфігурації та кодогенерації, STM32CubeIDE для розробки програмного забезпечення, а також підтримку для інших популярних IDE.

Широке застосування. STM32 застосовуються в різних областях, включаючи промисловість, автомобільну електроніку, медичні пристрої, консумерську електроніку, IoT-проекти та інші.

Наявність розвинутої екосистеми. STM32 користується великою та активною спільнотою розробників, яка допомагає обмінюватися досвідом, вирішувати проблеми та надавати корисні поради. Існує численні бібліотеки, приклади коду та різноманітні ресурси для полегшення розробки на платформі STM32.

Безпека та Захист від копіювання. Деякі моделі STM32 включають функціональність для захисту від копіювання (IP Protection) та захисту від атаки (Secure Boot) для забезпечення безпеки вбудованого програмного забезпечення.

Моделі з підтримкою Bluetooth та Wi-Fi. Деякі останні моделі STM32 мають вбудовану підтримку бездротових інтерфейсів, таких як Bluetooth і Wi-Fi, що розширює їх можливості для розробки IoT-проектів.

Довгий термін доступності. STM32 виробляються великим виробником STMicroelectronics, що забезпечує довгий термін доступності та підтримки для продуктів. Це важливо для проектів, які потребують стабільної поставки та продовженої підтримки.

Енергоефективність. Деякі моделі STM32 вирізняються високою енергоефективністю, що робить їх ідеальними для застосувань, де важлива тривалість роботи від батарей або вимоги до низького енергоспоживання.

STM32 визнані своєю гнучкістю, продуктивністю та надійністю, що робить їх популярними серед розробників усього світу для широкого спектру застосувань. Мікроконтролери STM32 широко використовуються у

промисловості, автомобільній промисловості, медичних пристроях, робототехніці, IoT-проектах та багатьох інших областях завдяки своїм високим технічним характеристикам та гнучкості.

STM32 є не лише технічно продуктивною платформою, але й платформою, що надає розробникам інструменти та можливості для реалізації різноманітних інноваційних проектів у своїх галузях. Наявність багатьох різних моделей дозволяє вибирати ті, які найкраще відповідають конкретним вимогам та обмеженням проекту.

Нижче представлено опис однієї з основоположних моделей сімейства STM32. STM32F103C8T6 – це одна з моделей мікроконтролерів з лінійки STM32F1 від STMicroelectronics. Ця модель є досить популярною серед розробників та дозволяє широко охопити специфіку проєктів від невеликих електронних пристроїв до більших промислових систем з урахуванням критичності, автомобільній електроніці тощо (рис. 2.2).

Основні технічні характеристики STM32F103C8T6:

– Ядро. Мікроконтролер базується на ядрі ARM Cortex-M3 з 32-розрядною архітектурою. Це ядро забезпечує високу продуктивність та ефективність в обробці сигналів.

– Частота тактового сигналу. STM32F103C8T6 може працювати на високій частоті, зазвичай до 72 МГц, що робить його досить потужним для багатьох застосувань.

– Обсяг флеш-пам'яті. Має 64 Кб вбудованої флеш-пам'яті для зберігання програмного коду.

– Оперативна пам'ять (RAM). Вбудована оперативна пам'ять об'ємом 20 Кб для забезпечення роботи програм.

– Периферійні пристрої. STM32F103C8T6 оснащений рядом периферійних пристроїв, включаючи GPIO (порти вводу/виводу), таймери, UART, SPI, I2C, ADC (аналогово-цифровий конвертер), та інші.

– Інтерфейси зовнішніх пристроїв. Має підтримку різноманітних інтерфейсів, таких як USB, CAN, інтерфейси для зовнішніх пам'ятей,

інтерфейси зв'язку.

– Розширення можливостей. STM32F103C8T6 має GPIO для з'єднання з різноманітними розширювальними платами, зокрема, може використовуватися з платами у форм-факторі Arduino.

– Наявність Bootloader. Деякі версії цього мікроконтролера мають вбудований загрузчик (bootloader), що дозволяє оновлювати програмне забезпечення через інтерфейс UART.

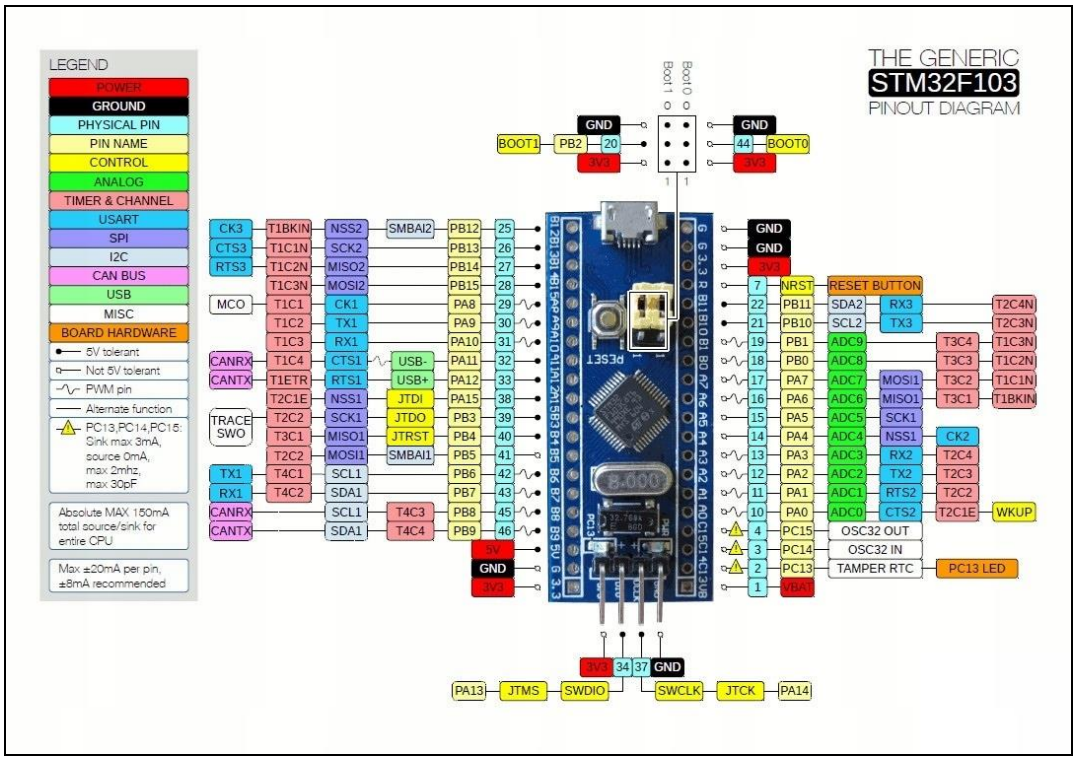

Рисунок 2.2 – Схематичне зображення плати розробки на базі мікроконтролера STM32F103C8T6

2.1.2 Сімейство плат ESP32

ESP-мікроконтролери – це сімейство мікроконтролерів, розроблених компанією Espressif Systems. Одним з найвідоміших представників цього сімейства є ESP8266 та ESP32. Ці МК відомі своєю високою продуктивністю, вбудованим Wi-Fi та Bluetooth, широким функціоналом, а також відмінною підтримкою від спільноти та розвитком open-source проектів (рис. 2.3).

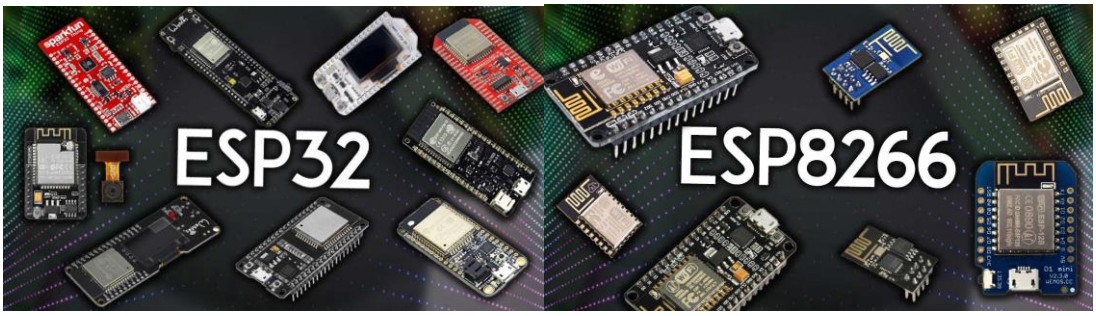

Рисунок 2.3 – Найпопулярніші моделі Espressif Systems

ESP32 – це високопродуктивний мікроконтролер, розроблений компанією Espressif Systems. Він є розвитком популярного ESP8266 і вирізняється рядом розширених можливостей та технічних характеристик.

Ось деякі ключові особливості ESP32.

Двоядерний процесор. ESP32 має два ядра – одне ядро (процесор Tensilica Xtensa LX6) використовується для виконання користувацького коду, а інше – для виконання системних задач. Це дозволяє взаємодіяти з різними задачами паралельно та забезпечує більшу продуктивність.

Wi-Fi та Bluetooth. ESP32 вбудовано підтримує Wi-Fi та Bluetooth. Це робить його ідеальним для розробки проектів IoT, які вимагають зв'язку з іншими пристроями чи хмарами.

Bluetooth Low Energy (BLE). Однією з ключових особливостей ESP32 є підтримка Bluetooth Low Energy, що використовується для з'єднань з енергозберігаючими пристроями, такими як сенсори, фітнес-трекери тощо.

Широкий спектр периферійних пристроїв. ESP32 має розширений набір периферійних пристроїв, таких як ADC (аналогово-цифровий конвертер), DAC (цифро-аналоговий конвертер), PWM (ширина імпульсу), I2C, SPI, UART, і багато інших. Це полегшує підключення до різних датчиків, вивід до дисплеїв тощо.

Можливість роботи в режимі точки доступу (AP mode). ESP32 може працювати в режимі точки доступу, що дозволяє підключати інші пристрої до його Wi-Fi мережі. Це дуже корисно для створення власних мереж або підключення інших пристроїв до ESP32.

Можливості енергозбереження. ESP32 підтримує різні режими енергозбереження, що робить його ідеальним для використання в проектах, які працюють від акумуляторів або потребують ефективного управління енергоспоживанням.

Зручне програмування. ESP32 можна програмувати за допомогою Arduino IDE, MicroPython, ESP-IDF (Espressif IoT Development Framework) та інших інтегрованих середовищ розробки. Це полегшує використання для розробників з різним досвідом.

Підтримка камери. Деякі моделі ESP32 мають вбудовану підтримку підключення камери, що відкриває можливості для проектів, пов'язаних із захопленням та обробкою зображень.

ESP32 широко використовується в різноманітних проектах, таких як IoT-пристрої, сенсорні системи, робототехніка, автоматизація будинку, електроніка для хобі тощо. Висока продуктивність та розширені можливості роблять ESP32 привабливим вибором для багатьох розробників (рис. 2.4).

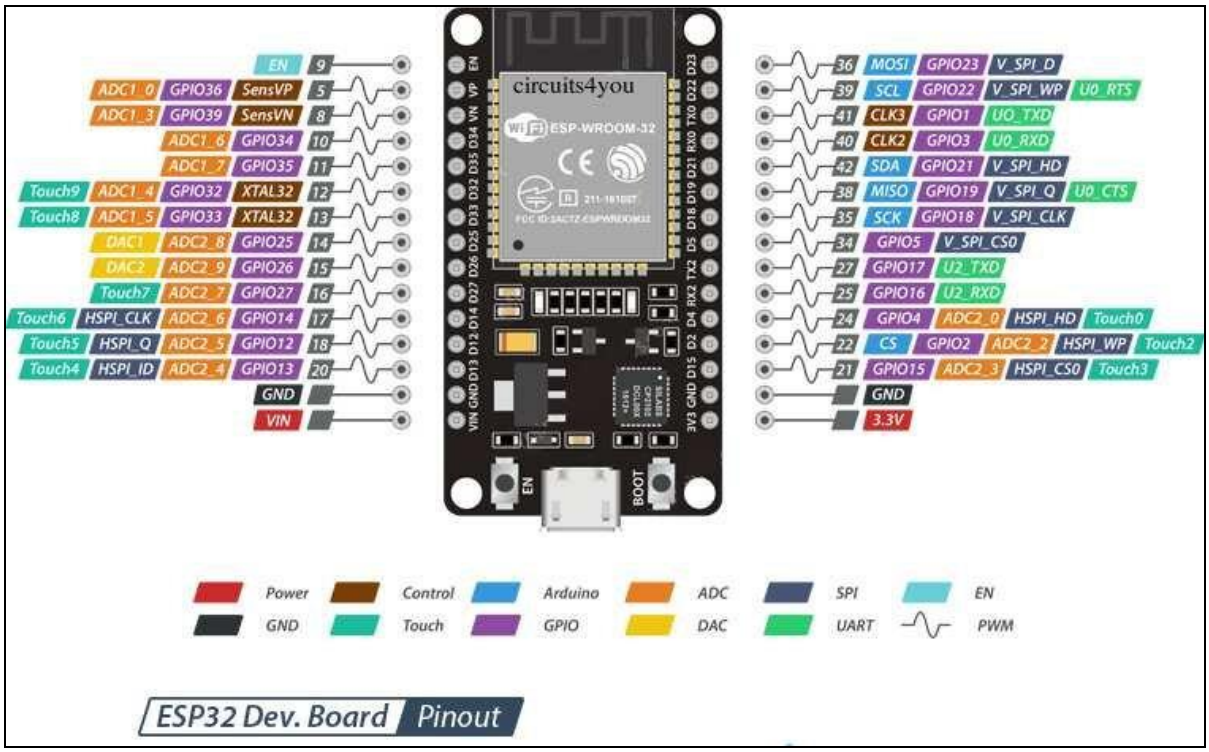

Рисунок 2.4 – Плата розробки ESP-WROOM-32

Характеризуючі різні сімейства мікроконтролерів, можна прийти до однозначного висновку, що кожне сімейство має свою спеціалізацію або ж є взаємозамінними. Мають додаткові протоколи передачі даних чи навпаки створені для енергоефективних імплементаційних рішень і обмежуються 1-2 протоколами з низькою швидкістю і, відповідно, є найменш ресурсовитратними. Наприклад сімейства мікроконтролерів ESP популярні за рахунок недорогої імплементації бездротових протоколів передачі даних. А сімейства мікроконтролерів компанії Microchip (PIC) досягають високоархітектурних рішень стандартними 8-бітними мікроконтролерами, використовуючи ті ж самі апаратні реалізації протоколів передачі даних, периферії тощо.

Основний опис, характеристики та відмінності протоколів передачі даних, що використані у кваліфікаційній роботі, наведений у підрозділі 2.2.

2.2 Протоколи передачі даних

2.2.1 Універсальний Асинхронний Приймач/Передавач

UART (Universal Asynchronous Receiver/Transmitter) є популярним протоколом передачі даних, який використовується для забезпечення зв'язку між двома пристроями через послідовний інтерфейс. Назва "асинхронний" походить від того, що немає окремого тактового сигналу, який координує передачу та прийом даних між пристроями.

Основні характеристики UART наступні.

Лінії зв'язку. UART використовує дві лінії для забезпечення двонаправленого обміну даними: TX (Transmit): лінія для передачі даних; RX (Receive): лінія для прийому даних.

Стартовий біт (Start Bit). Це завжди перший біт у кадрі даних і вказує на початок передачі. Його значення завжди нуль, і він розпочинає процес читання/запису даних.

Дані (Data Bits). Кількість бітів, що кодують самі дані. Зазвичай використовуються 8 бітів, але іноді може бути використано 7 або 9 бітів, залежно від конфігурації. Стоповий біт (Stop Bit) вказує на завершення передачі даних. Зазвичай використовується один або два стопові біти.

Біти парності (Parity Bits). Опційний елемент, який може додаватися до кадру для виявлення помилок. Може бути парним, непарним або відсутнім (без бітів парності).

Швидкість передачі (Baud Rate). Визначає кількість бітів, які передаються або приймаються за одну секунду. Швидкість вимірюється в бодах (бітах на секунду).

Процес передачі даних у UART виглядає наступним чином:

– передача починається зі стартового біта, який має значення "0";

– дані (8 бітів або інше, залежно від налаштувань) відправляються послідовно, молодший біт перший;

– опційний біт парності додається, якщо він використовується;

– один або кілька стопових бітів завершують передачу;

– прискорювач використовується для визначення швидкості передачі.

UART є досить простим та ефективним способом передачі даних і використовується в широкому спектрі пристроїв, від мікроконтролерів та периферійних пристроїв до комп'ютерів та зовнішніх пристроїв (рис. 2.5).

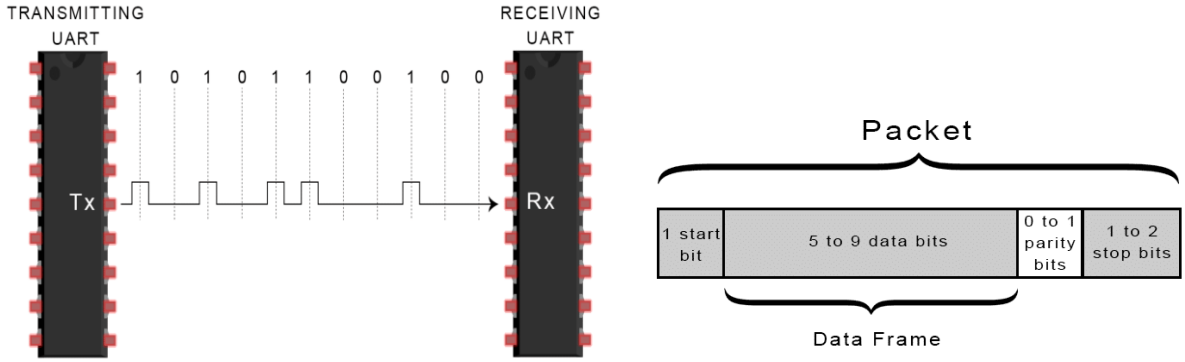

Рисунок 2.5 – Передача даних за протоколом UART

2.2.2 Універсальний Синхронний/Асинхронний Приймач/Передавач

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) представляє собою розширену версію протоколу UART, яка підтримує і синхронний, і асинхронний режими передачі даних. USART дозволяє використовувати окремий тактовий сигнал для синхронізації передачі даних, що дозволяє досягти більш високих швидкостей передачі та забезпечити точнішу синхронізацію між пристроями.

Основні характеристики USARTнаступні.

Асинхронний режим (UART): У цьому режимі відсутній окремий тактовий сигнал, і передача даних відбувається за допомогою стартового біта, бітів даних, бітів парності (опційно) та стопового біта.

Синхронний режим. USART може працювати в режимі, де для синхронізації використовується окремий тактовий сигнал. У цьому режимі відсутній стартовий біт, і дані передаються відправником та приймачем в залежності від тактового сигналу.

Біти даних, біти парності і стопові біти. Аналогічно UART, USART підтримує налаштування кількості бітів даних, бітів парності (якщо вони використовуються) та стопових бітів.

Швидкість передачі (Baud Rate). USART може працювати на високих швидкостях, забезпечуючи ефективний обмін даними. Швидкість вимірюється в бодах (бітах на секунду).

Підтримка повного дуплексу. USART дозволяє одночасну передачу та прийом даних, що робить його ідеальним для повного дуплексу.

Можливість генерації тактового сигналу. USART може генерувати свій власний тактовий сигнал або працювати із зовнішнім тактовим сигналом у синхронному режимі.

USART використовується в різних застосуваннях, таких як мікроконтролери, модеми, передавачі-приймачі, вбудовані системи та інші пристрої, які вимагають надійного обміну даними (рис. 2.6).

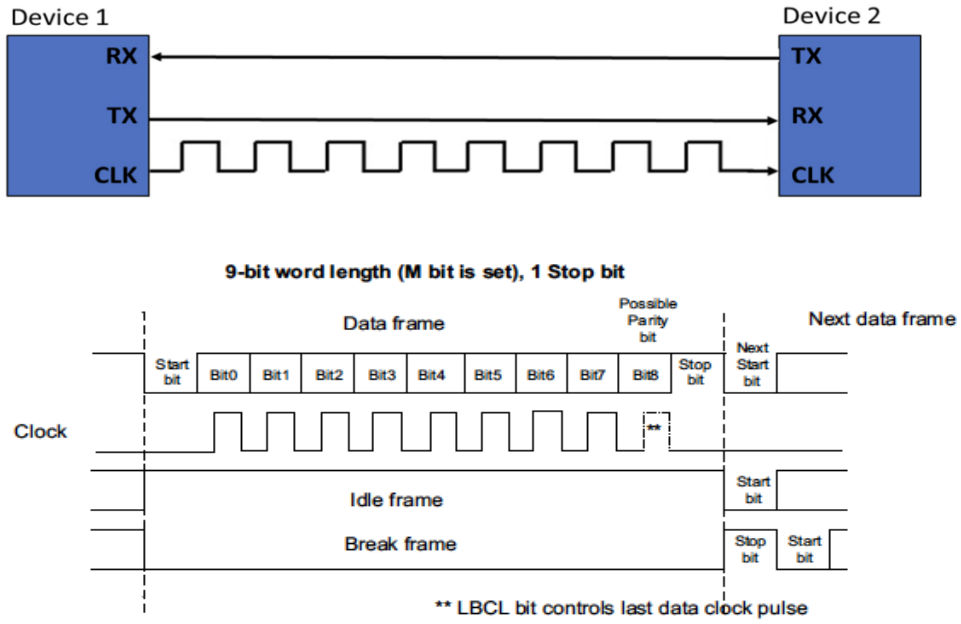

Рисунок 2.6 – Протокол роботи USART

2.2.3 Послідовний периферійний інтерфейс

SPI (Serial Peripheral Interface) - це протокол передачі даних, який дозволяє обмінювати інформацією між мікроконтролерами та периферійними пристроями або між двома мікроконтролерами. Цей протокол використовує окремі лінії для передачі даних, тактового сигналу та сигналів управління.

Основні характеристики протоколу SPI наступні.

Лінії зв'язку:

– MISO (Master In Slave Out) – лінія для передачі даних від пристрою-підлеглого до майстра;

– MOSI (Master Out Slave In) – лінія для передачі даних від майстра до пристрою-підлеглого;

– SCLK (Serial Clock) – тактовий сигнал, який визначає частоту передачі даних;

– SS/CS (Slave Select/Chip Select): Сигнал вибору пристрою-раба (декілька пристроїв можуть бути підключені до одного майстра).

Режими передачі. SPI підтримує кілька режимів передачі, де визначаються фаза та положення тактового сигналу. SPI підтримує повний дуплекс, що означає можливість одночасної передачі та прийому даних.

Ведучий та ведений. В протоколі SPI є один ведучий (зазвичай мікроконтролер) та один або декілька пристроїв-ведених (наприклад, датчики, дисплеї, пам'яті тощо). Кількість бітів в кожному слові може бути налаштована.

Робоча частота. Частота тактового сигналу може бути встановлена заздалегідь, щоб визначити швидкість передачі даних. Це дозволяє налаштовувати швидкість зв'язку в межах можливостей обладнання. Швидкість передачі. SPI дозволяє високі швидкості передачі даних, що робить його відмінним для високошвидкісних застосувань.

Протокол SPI використовується в різних областях, таких як мікроконтролерні системи, комунікації з периферійними пристроями, обробка сигналів, аудіо- та відеоапаратура, і багато іншого (рис. 2.7).

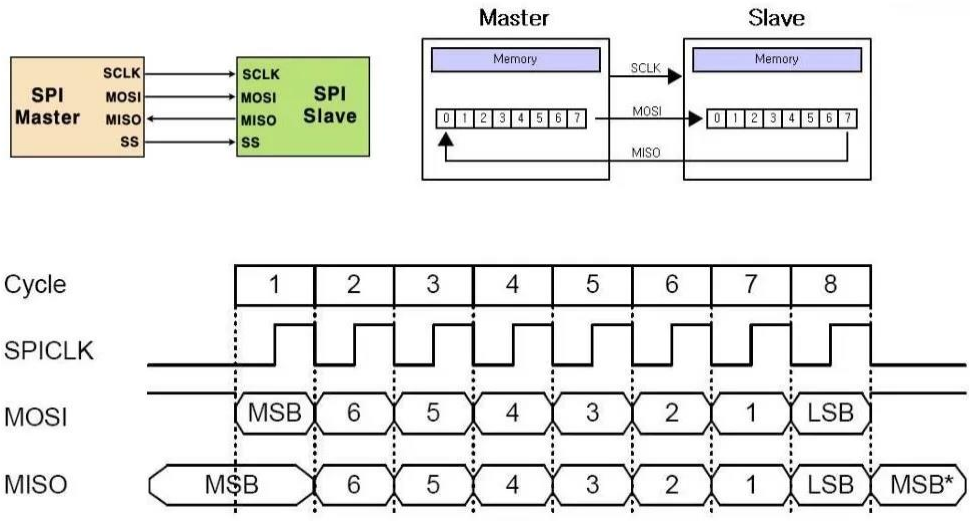

Рисунок 2.7 – Протокол SPI

2.2.4 Інтерфейсна шина

I^2C (Inter-Integrated Circuit) – це протокол передачі даних, який був розроблений фірмою Philips (тепер NXP Semiconductors) для забезпечення

зв'язку між різними пристроями на платі чи в одній системі. I^2C є дводротовим протоколом, який використовує пару ліній для комунікації між пристроями: лінія даних (SDA – Serial Data Line) та лінія такту (SCL - Serial Clock Line).

Основні характеристики протоколу I^2C наступні.

Дводротовий інтерфейс. I^2C використовує всього дві лінії для передачі даних та синхронізації – SDA та SCL. Це робить його економічним і придатним для застосувань, де обмежені ресурси.

Архітектура ”Ведучий-ведений”: I^2C працює за принципом "Ведучий - ведений ". Ведучий – це пристрій, який ініціює обмін даними, а ведений – це пристрій, який відповідає на запити майстра.

Адресація пристроїв. Кожен I^2C-пристрій має унікальну семибітну або дев'ятибітну адресу, що визначає його місце в мережі. Адреса 0x00 зазвичай резервується для загально використовуваних адрес.

Стартовий та стоповий біти. Передача даних починається зі стартового біта, який вказує на початок комунікації. Сесія передачі завершується стоповим бітом. Це допомагає визначити межі кожного пакету даних.

Байтова орієнтованість. Дані передаються в байтах, і після кожного байта може бути біт підтвердження (ACK) чи негативне підтвердження (NACK), що дозволяє визначити успішність передачі даних.

Мульти-майстер режим. Протокол I^2C підтримує можливість мульти-мастера, коли більше одного пристрою може виступати в ролі майстра. Це робить його гнучким для використання в системах з декількома пристроями.

Швидкість передачі. I^2C підтримує різні швидкості передачі, такі як Standard (100 kbit/s), Fast (400 kbit/s), та High Speed (3.4 Mbit/s).

Протокол I^2C широко використовується в електроніці для підключення різних пристроїв, таких як сенсори, EEPROM-пам'ять, датчики температури, часу реального, та інші, до мікроконтролерів чи інших пристроїв.

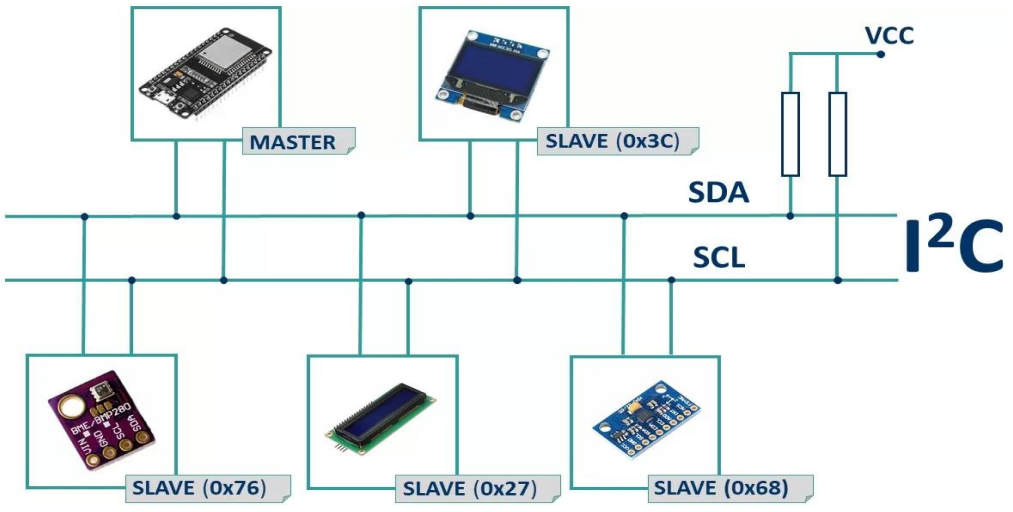

Рисунок 2.8 – Інтерфейс I^2C

2.2.5 Локальна мережа контролерів

Протокол передачі даних CAN (Controller Area Network) розроблений для надійної передачі даних в мережах автомобільної промисловості, але його застосування розширилося і на інші області, де важлива висока надійність і масштабованість передачі даних між пристроями.

Основні характеристики протоколу CAN наступні.

Архітектура. CAN використовує архітектуру мережі з контролерами, які можуть взаємодіяти один з одним. В мережі може бути кілька контролерів, і кожен з них може надсилати чи отримувати дані.

Двоспособова передача даних. CAN підтримує двоспособову передачу даних між пристроями. Контролер може надсилати дані та отримувати дані від інших контролерів в мережі.

Фізичний рівень. Фізичний рівень CAN може використовувати два типи мережевого кабеля: вита пара і оптичний кабель. Вита пара часто використовується для коротких відстаней, тоді як оптичний кабель може використовуватися для великих відстаней чи в умовах, де є електричні перешкоди.

Швидкість передачі даних. CAN підтримує різні рівні швидкості

передачі даних, які можуть бути налаштовані залежно від конкретних вимог системи. Швидкості можуть варіюватися від декількох Кбіт до Мбіт/сек.

Фрейми. Дані в CAN передаються у вигляді фреймів, які можуть бути стандартними або розширеними. Стандартні фрейми мають 11 біт для ідентифікатора, тоді як розширені фрейми мають 29 біт. CAN дозволяє встановлювати маски та фільтри для ідентифікації та фільтрації фреймів. Це дозволяє певним контролерам отримувати тільки ті дані, які їм потрібні.

Контроль передачі. CAN включає механізми для виявлення та корекції помилок в передачі даних. Це забезпечує високу надійність передачі даних навіть в умовах високого рівня електромагнітних перешкод.

Керування конфліктами. Механізми розв'язання конфліктів у випадку спільної передачі можуть забезпечити високу ефективність мережі, дозволяючи багатьом пристроям взаємодіяти без зайвих колізій.

Протокол CAN знайшов широке застосування в автомобільній промисловості, промислових системах автоматизації та інших областях, де необхідна надійна та масштабована система передачі даних.

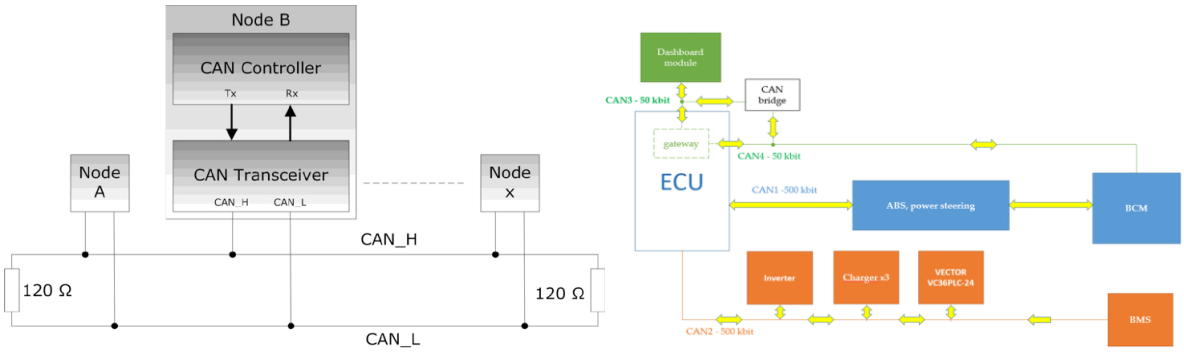

Рисунок 2.9 – CAN шина та її основне використання

Протоколи, опис яких наведено в даному розділі, використовуються у різних сценаріях та мають свої властивості та переваги. Вибір конкретного протоколу залежить від конкретних потреб та вимог як проєкту, так і архітектурного рішення взаємообміну даними між ключовими активними

вузлами системи. Відмінності даних протоколів та їх імплементація на певному сімействі буде висвітлена та обгрунтована.

Існує ще один протокол передачі даних в автомобільній промисловості LIN (Local Interconnect Network), який можна зазначити для довідки та більшої порівняльної характеристики. Даний протокол використовує “проміжну шину” та дозволяє розвантажити обробку, аналіз та передачу даних до блоку управління двигуном автомобіля, яка здійснюється протоколом CAN.

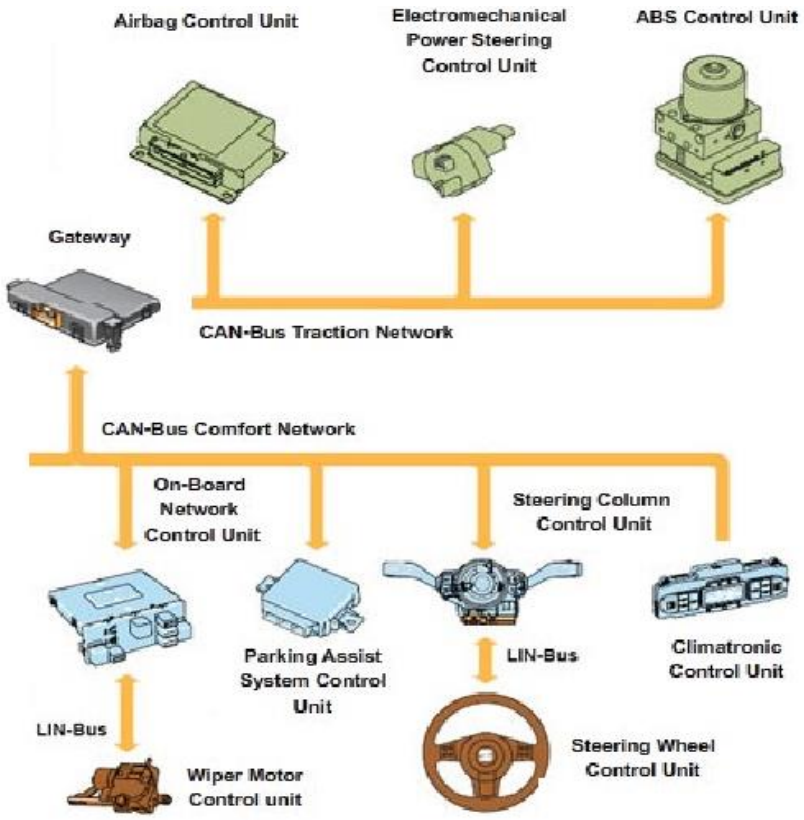

Рисунок 3.х - Використання LIN-інтерфейсу

Основні властивості LIN наступні :

– проста мережа з майстером і веденими, які підключені за допомогою одного проводу (лінії), майстер ініціює передачу даних, але немає строгої синхронізації.

– лінійна адресація, кожен пристрій у мережі має свою унікальну адресу, зазвичай не використовується біти парності.

– швидкість нижча порівняно з іншими протоколами, призначена для низькошвидкісних мереж, таких як у внутрішніх системах автомобілів.

– зазвичай використовується в автомобільній промисловості для низькошвидкісної комунікації між різними пристроями у внутрішніх системах автомобіля, таких як двері, вікна, світло тощо.

2.3 Інтегрований перетворювач інтерфейсів FT2232

FT2232 – це інтегрований пристрій від компанії FTDI (Future Technology Devices International), який використовується для обробки та інтерфейсування з USB. Зазвичай використовується як інтерфейс USB-UART (Universal Asynchronous Receiver/Transmitter) або USB-JTAG (Joint Test Action Group), та має два канали (A та B), кожен з яких може працювати незалежно. FT2232 широко використовується в електроніці, робототехніці та розробці вбудованих систем для забезпечення зручного з'єднання та обміну даними між комп'ютером та різними пристроями.

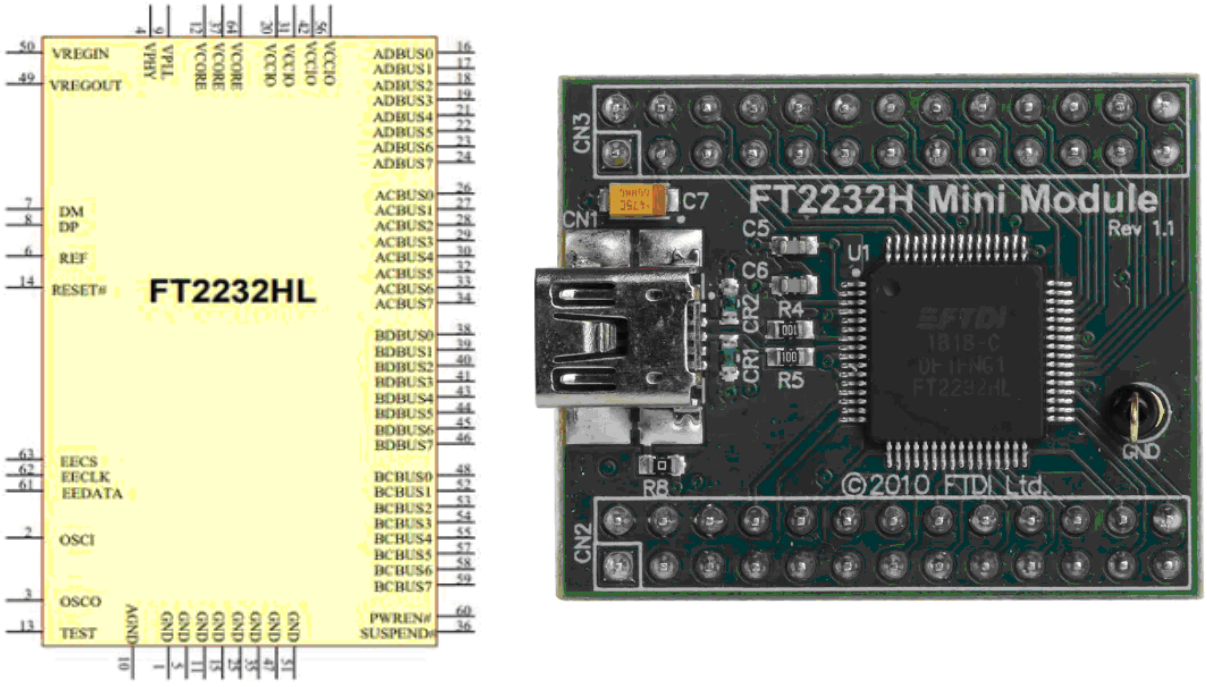

Рисунок 2.10 – Плата FT2232HL та її розпіновка

Основні характеристики FT2232 наступні.

USB-UART конвертер. FT2232 часто використовується як пристрій USB-UART для перетворення даних між USB і серійним (UART) інтерфейсом. Це зручно для зв'язку між комп'ютером та мікроконтролером чи іншими пристроями через USB-порт. Також UART використовується для реалізації інтерфейсу для обміну даними з іншими пристроями чи мікроконтролерами через серійний порт.

USB-JTAG конвертер. Якщо FT2232 використовується як JTAG-програматор, то він має вбудований контролер JTAG для взаємодії з вбудованими системами, які підтримують JTAG. Окрім цього, FT2232 може використовуватися як USB-JTAG конвертер для програмування та відладки вбудованих систем, таких як мікроконтролери чи FPGA (Field-Programmable Gate Array), що підтримують JTAG інтерфейс.

Два незалежних канали. FT2232 має два канали (Channel A та Channel B), які можуть функціонувати незалежно. Це означає, що пристрій може обробляти два різних потоки даних одночасно.

Швидкість передачі даних. Підтримує широкий спектр швидкостей передачі даних, включаючи стандартні швидкості UART (від 300 біт/с до 12 Мбіт/с) та USB 2.0 швидкості (до 480 Мбіт/с).

Широкий діапазон напруги. Має високу сумісність з різними рівнями напруги (від 1.8V до 5V), що дозволяє використовувати його з різними логічними рівнями у вбудованих системах

Різні режими роботи. FT2232 підтримує різні режими роботи, такі як Bit-Bang режим (для програмованого виводу даних) та MPSSE (Multi-Protocol Synchronous Serial Engine) режим для використання в різних пристроях.

Підтримка різних операційних систем. Сумісний з різними операційними системами, включаючи Windows, Linux, MacOS, що робить його універсальним для різних розробників.

Широкі можливості відлагодження. FT2232 може бути використаний для відладки вбудованих систем через JTAG інтерфейс. Це робить його

корисним інструментом для розробників, які потребують відслідковувати та відлагоджувати програми на низькому рівні.

Велика спільнота та підтримка. Як із загальністю продуктів від FTDI, FT2232 користується підтримкою активної спільноти розробників. Це означає наявність багатьох ресурсів, допомоги та документації для користувачів.

Простота використання. Інтерфейс USB дозволяє легко підключати FT2232 до комп'ютера, що забезпечує швидкий доступ до введення-виведення інших пристроїв. Крім того, наявність драйверів для різних операційних систем спрощує процес налаштування та використання.

Bit-Bang режим. У режимі Bit-Bang, FT2232 може виступати як програмований цифровий вивід/вхід, що дозволяє гнучко використовувати його для різних завдань введення-виведення. Тобто і даному режимі піни плати можуть бути 100% запрограмовані за сценарієм розробника або користувача, що повністю покриває

Можливості MPSSE режиму. У режимі MPSSE, FT2232 може функціонувати як мультіпротокольний синхронний двигун для підтримки різних протоколів, таких як SPI, I2C, JTAG, і UART. Це забезпечує йому широкий спектр можливостей.

FT2232 є корисним інструментом для розробників, що працюють у сфері вбудованих систем та електроніки. Він дозволяє забезпечити зв'язок та відлагодження програм на різних етапах розробки проектів.

Нижче представлений поверхневий опис будови плати.

В FT2232 використовується вбудоване ядро або процесор для виконання внутрішніх операцій, обробки даних та керування іншими функціональними блоками. Контролер USB відповідає за взаємодію з USB інтерфейсом комп'ютера. Цей блок забезпечує відправку та отримання даних через USB, використовуючи стандартні протоколи USB.

Деякі FT2232 мають вбудовану флеш-пам'ять або йому потрібна зовнішня пам'ять для зберігання програмного забезпечення, конфігураційних

даних та інших важливих інформаційних блоків. Генератор тактових сигналів (Clock Generator) відповідає за створення тактових сигналів, необхідних для синхронізації роботи всіх інших компонентів пристрою. Виводи загального призначення GPIO (General Purpose Input/Output). дозволяють підключати FT2232 до різних зовнішніх пристроїв чи датчиків. Вони можуть використовуватися для виводу сигналів або для зчитування зовнішніх сигналів. FT2232 використовує драйвери та програмне забезпечення для взаємодії з комп'ютером та забезпечення правильної роботи пристрою. Це включає в себе драйвери для різних операційних систем, а також бібліотеки для розробки програмного забезпечення.

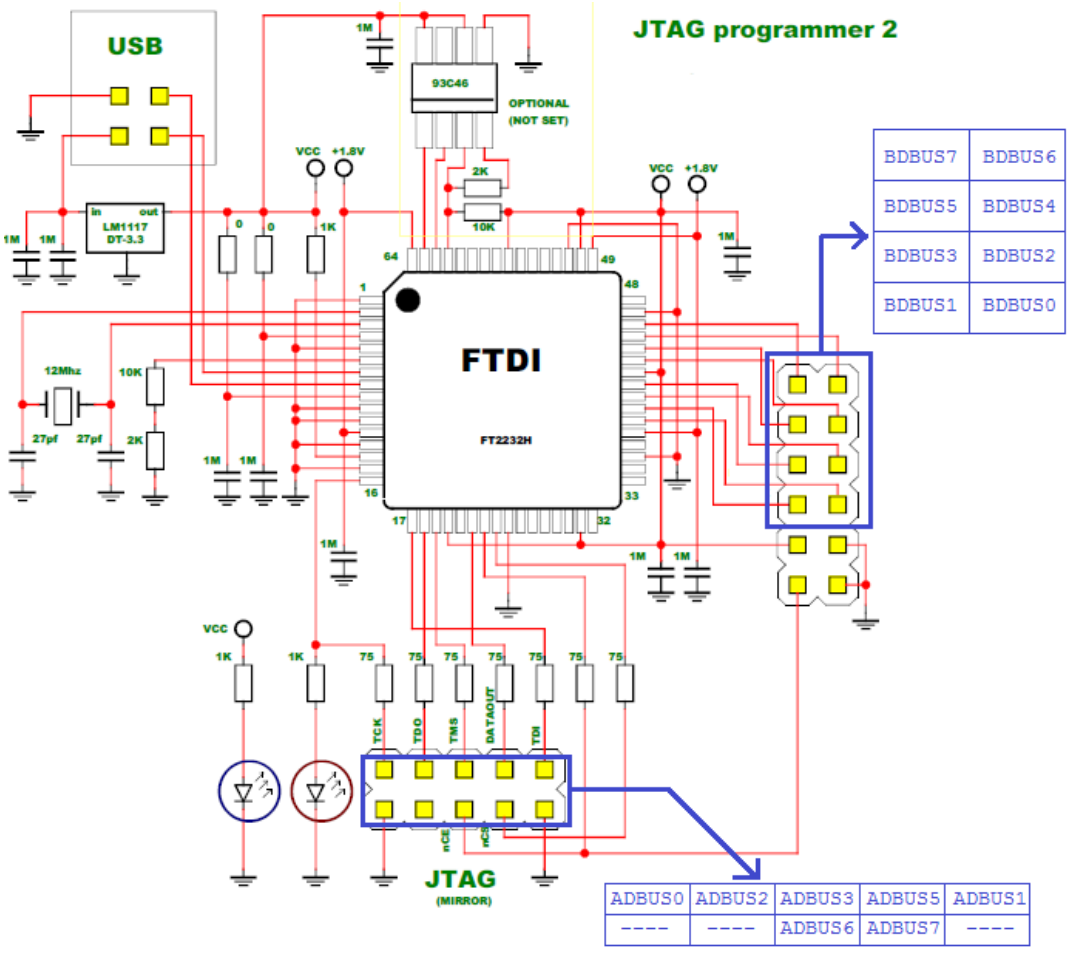

Рисунок 2.11 – Схематичне зображення використання Bit-Bang FT2232

В інтегрованому пристрої такого класу також можуть бути інші компоненти, такі як регулятори напруги, інтерфейси для підключення

зовнішніх джерел живлення, захист від перевантаження та інші, залежно від конкретної моделі та призначення.

FT2232 має різні піни, які можуть використовуватися для різних цілей в залежності від режиму роботи та конфігурації пристрою. Варто зазначити, що конкретні функції пінів можуть змінюватися залежно від режиму роботи та конфігурації FT2232. Важливо користуватися документацією, яка відповідає конкретній моделі пристрою.

2.4 Модель мікроконтролерної системи імітаційного діагностування

Імітаційне налагодження та діагностування мікроконтролерних систем пов'язано з вибором необхідних компонентів серед усіх методів і засобів, що застосовуються до мікроконтролерних пристроїв. Тому комплекс заходів у таких системах складається з іншого набору інструментів, які можна виділити в спеціалізовану архітектуру. Виділяється основні рівні імітації при діагностуванні: інтерфейсний, апаратний (процесорний) і програмний рівні.

При тестовому діагностуванні мікроконтролерних систем вважається, що сам МК (тобто апаратна частина системи) не має дефектів за визначенням. Це реалізується шляхом вхідного (вихідного) контролю при виробництві МК. Тестування програмної частини здійснюється, як правило, позитивними тестами замовника на рівні верифікації за правилами тестування програмного забезпечення.

Таким чином найбільш важливою частиною при налагодженні мікроконтролерної системи взаємодії з периферією є інтерфейсний рівень налагодження.

UART (Universal Asynchronous Receiver/Transmitter), USART (Universal Synchronous/Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), CAN (Controller Area Network), та LIN (Local Interconnect Network) – це різні протоколи для передачі даних між

пристроями. Вони мають свої унікальні характеристики та застосування. В таблиці 2.1 наведена їх порівняльна характеристика.

Таблиця 2.1 – Порівняльні характеристики інтерфейсів

	Протоколи				
	UART	USART	SPI	I^2C	CAN
Архітек-тура	Двоспосо-бова асинхронна	Двоспосо-бова	Ведучий - ведений з 4 лініями	Ведучий - ведений з 2 лініями	Ведучий - ведений з 2 лініями
Лінії зв'язку	TX (Transmit) та RX (Receive).	TX (Transmit) та RX (Receive).	MOSI, MISO, SCLK, та SS/CS	SDA та SCL	CAN_H та CAN_L
Синхроні-зація	Асинхронна	Синхронна і асинхронна	Синхронна	Синхронна	Синхронна
Адресація	Відсутня	Відсутня	Кожен пристрій адресується	Кожен пристрій адресується	Кожен пристрій адресується
Біти парності	Підтримує	Підтримує	Не підтримує	Не підтримує	CRC
Швидкість передачі даних	Налашто-вується	Налашто-вується	Налашто-вується	Налашто-вується	Налашто-вується для СРЧ
Застосуван ня	Загального призначення	Загального призначення	Для швидкої передачі даних МК та периферії	Для швид-кої передачі даних між адресами в мережах	В автомобі-лях для передачі даних між системами

Таким чином, виходячи з різноманітності властивостей протоколів та галузей їх застосування доцільно для системи імітаційного налагодження застосовувати «універсальний» пристрій, який підтримує усі зазначені протоколи. Це перетворювач інтерфейсів FT2232HL-Board.

При цьому в якості тестера виступає одноплатний ПК, який через інтерфейс USB підєднується до плати FT2232HL-Board. На тестері реалізуються тестові програми, які діагностують взаємодію МК з периферійними пристроями. Структура інтерфейсної частини МК системи імітаційного діагностування наведена на рис.2.12.

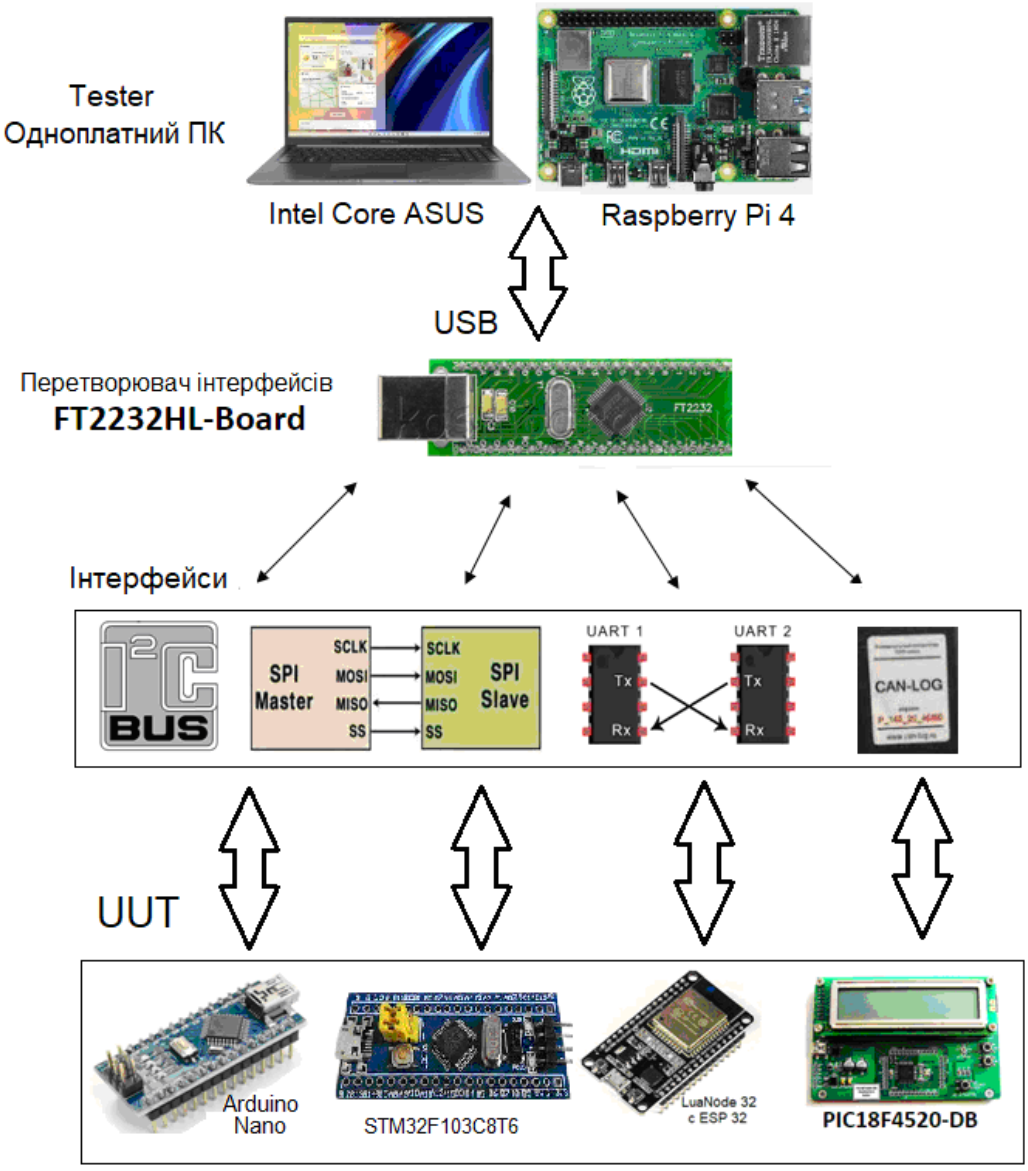

Рисунок 2.12 – Структура інтерфейсної частини мікроконтролерної системи імітаційного діагностування

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДІАГНОСТИЧНОГО ЕКСПЕРИМЕНТУ

3.1 Пристрій логічного керування дорожнім світлофором

Для перевірки запропонованого метода мікроконтролерних моделей часових автоматів розглянемо систему управління дорожнім світлофором з реалізацією функції «зелена хвиля» (Gw) для автомобілів спеціальних служб. Якщо на дорозі горить червоне світло світлофора, а автомобіль спеціальних служб (швидка допомога, ДСНС, військові) потребує позачергового проїзду, то він "подає" сигнал Gw і, відповідно, "вмикає" зелене світло на своєму маршруті. Відмітимо, що на перехресті, де встановлюється зазначений світлофор, не передбачений пішохідний перехід.

Розглянемо алгоритм функціонування дорожнього світлофора, який працює у двох режимах: денному і нічному.

Множина вхідних сигналів X = {Onn, St, Gw}, де Onn = {0, 1} – сигнал включення дорожнього світлофора та запуску нічного циклу, St = {0, 1} – сигнал запуску денного циклу роботи світлофора, Gw = {0, 1} – сигнал для включення зеленого світла у режимі «зелена хвиля». Таким чином, Onn і St є оповіщувальними сигналами, а Gw - подією.

Множина вихідних сигналів для світлофорів Y = {R, YRG, YGR, G}, де R1 – сигнал включення червоного світла на основній дорозі, G – сигнал включення зеленого світла світлофора на дорозі, YRG – сигнали включення жовтого світла на дорозі (на зміні R – G), YGR – сигнали включення жовтого світла на дорозі (на зміні G – R), Відмітимо, що $\forall R \to \overline{G}$.

Інтерфейс системи управління дорожнім світлофором представлений на рисунку 3.1.

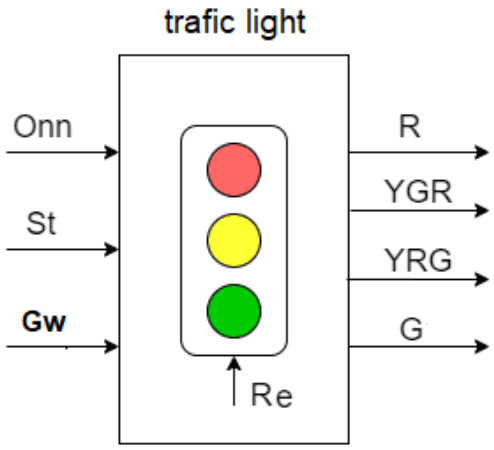

Рисунок 3.1 – Інтерфейс системи управління дорожнім світлофором

Визначимо стани керуючого автомата :

- стан a1 –включення автомата, вихідних сигналів немає, затримка to_1;
- стан a2 –жовтий при зміні (G–R) , виходи {YGR, R}, затримка to_2;
- стан a3 – червоний на дорозі, виходи {R}, затримка to_3;
- стан a4 – жовтий при зміні (R–G) , виходи {YRG, R}, затримка to_2;
- стан a5 – зелений на дорозі, виходи { G,}, затримка to_3;
- стан a6– включення зеленого на дорозі по сигналу Gw, затримка to_6;
- стан a7 – горить тільки жовтий, затримка to_1.

Використовується внутрішній керуючий сигнал Re, який визначає режим повернення зі стану a6 після закінчення затримки to_6. Визначимо t_{gw} як поточний час включення режиму Gw в проміжок часу to_3, при цьому $to_6 < to_3$. Таким чином, Re=0, якщо $(t_{gw} + to_6) < to_3$ та Re=1, якщо $(t_{gw} + to_6) >= to_3$.

Алгоритм роботи світлофора наступний. При включенні пристрою керування (Onn=1) запускається нічний цикл роботи світлофора, відбувається миготіння жовтого світла на основній дорозі ($a_1 – a_7$). Після запуску денного циклу (St=1) реалізується така система переходів ($a_2 – a_3 – a_4 – a_5 – a_2$). У стані a_3, коли на дорозі горить червоний, визначено вікно прийому $t_c = to_3$ для зовнішньої події Gw (включення сигналу «зелена хвиля»). При обробці даної події керуючий автомат переходить в стан a_6. При цьому на дорозі загоряється зелений і горить протягом to_6 . За період t_c може бути прийнятий

тільки один сигнал (зовнішня подія) Gw. Якщо час to_6 закінчується до закінчення to_3, то світлофор повертається до стану a3 (червоне світло), в іншому випадку після закінчення to_6 загоряється жовте світло і світлофор працює в стандартному денному режимі.

Таким чином, для системи управління дорожнім світлофором може бути побудований темпоральний граф переходів часового автомата Мура (рис. 3.2).

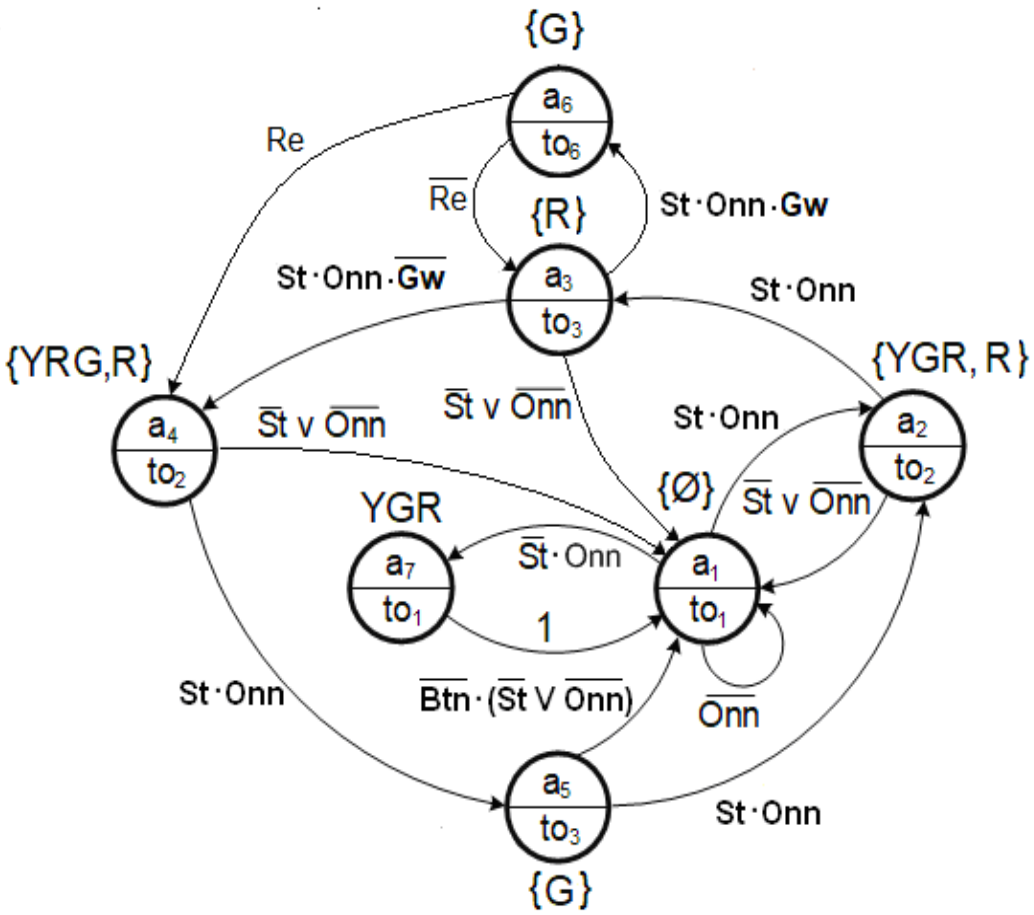

Рисунок 3.2 – Граф переходів автомата Мура для системи управління дорожнім світлофором

Часова діаграма функціонування дорожнього світлофора с функцією «зелена хвиля» (Gw) наведена на рис.3.3.

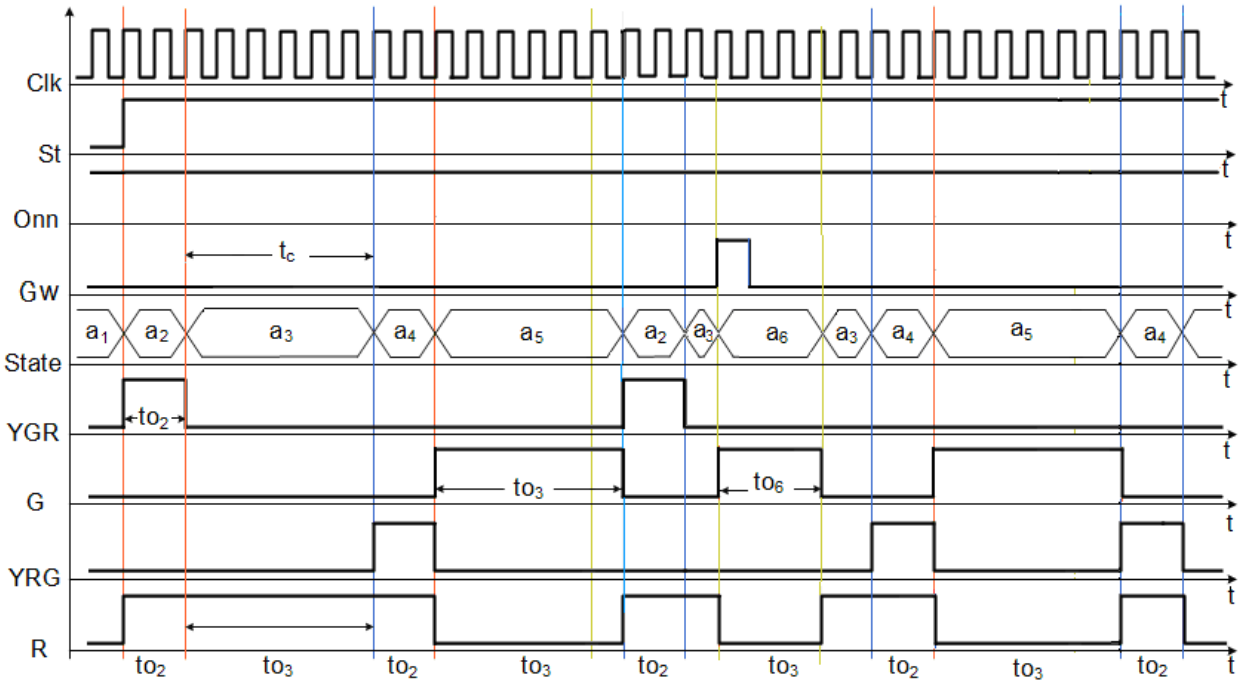

Рисунок 3.3 – Часова діаграма автомата Мура для системи управління дорожнім світлофором

3.2 Програмне забезпечення роботи інтерфейсів

Розглянемо програмування плати реалізації інтерфейсів FT2232 на демо-прикладі передачі повідомлення з FT2232 до Arduino Nano за допомогою протоколу UART.

Для взаємодії з FT2232 через UART, використаємо дві мови програмування, такі як Python та C++.

При імплементації мовою Python, необхідно використати бібліотеку PySerial для створення інтерфейсу UART. При імплементації мовою C++ - SerialStream для OS Windows та libftdi для OS Linux. Обидві бібліотеки потребують мануального встановлення. Встановити бібліотеки можна звернувшись до офіційної документації PySerial, Boost (SerialStream) та libftdi відповідно. Підключення вказаних бібліотек та протоколів для різних операційних систем наведено на рис.3.4, 3.5, 3.6.

```python
import serial

#OS Windows
# Ініціалізація об'єкту Serial для взаємодії з FT2232
serial_if = serial.Serial('COMx', 9600)  # COMx - вказати номер COM-порту та встановити швидкість передачі (бод)

#OS Linux
# Ініціалізація об'єкту Serial для взаємодії з FT2232
serial_if = serial.Serial('/dev/ttyUSBx', 9600)  # /dev/ttyUSBx - вказати номер USB девайсу та встановити швидкість передачі (бод)

try:
    # Відправлення даних FT2232
    serial_if.write(b'Hello From FT2232!\n')

    # Отримання та виведення даних від Arduino
    received_data = serial_if.readline()
    print(f'Received data: {received_data.decode()}')
except Exception as e:
    print(f'Error: {str(e)}')
finally:
    # Закриття Serial порту
    serial_if.close()
```

Рисунок 3.4 – Приклад передачі даних з FT2232 мовою Python

```cpp
#include <iostream>
#include <SerialStream.h>

int main()
{

    // OS Windows Вказати порт та швидкість передачі даних
    LibSerial::SerialStream serial("COMx", LibSerial::SerialStreamBuf::baud_rate(9600));

    try
    {
        // Відкриття порту
        if (serial.IsOpen())
        {
            // Відправлення даних до Arduino Nano
            serial << "Hello From FT2232 C++!" << std::endl;

            // Отримання та виведення даних від Arduino Nano
            std::string received_data;
            std::getline(serial, received_data);
            std::cout << "Received data: " << received_data << std::endl;
        }
        else
        {
            std::cerr << "Error opening serial port." << std::endl;
        }
    }
    catch (const std::exception &e)
    {
        std::cerr << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

Рисунок 3.5 – Приклад передачі даних з FT2232 мовою C++ на OS Windows

```c
#include <stdio.h>
#include <stdlib.h>
#include <ftdi.h>

int main() {
    struct ftdi_context *ftdi;
    int ret;

    // Ініціалізація контексту FTDI
    if ((ftdi = ftdi_new()) == 0) {
        fprintf(stderr, "Failed to initialize FTDI context\n");
        return EXIT_FAILURE;
    }

    // Відкриття пристрою за допомогою Vendor ID та Product ID FT2232
    if ((ret = ftdi_usb_open(ftdi, 0x0403, 0x6010)) < 0) {
        fprintf(stderr, "Can't open FTDI device: %d (%s)\n", ret, ftdi_get_error_string(ftdi));
        ftdi_free(ftdi);
        return EXIT_FAILURE;
    }

    // Налаштування параметрів UART
    if ((ret = ftdi_set_baudrate(ftdi, 9600)) < 0) {
        fprintf(stderr, "Can't set baudrate: %d (%s)\n", ret, ftdi_get_error_string(ftdi));
        ftdi_usb_close(ftdi);
        ftdi_free(ftdi);
        return EXIT_FAILURE;
    }

    // Відправлення даних через UART
    const char data[] = "Hello, STM32!";
    if ((ret = ftdi_write_data(ftdi, (unsigned char *)data, sizeof(data))) < 0) {
        fprintf(stderr, "Failed to write data: %d (%s)\n", ret, ftdi_get_error_string(ftdi));
    }

    // Закриття пристрою
    ftdi_usb_close(ftdi);
    ftdi_free(ftdi);

    return EXIT_SUCCESS;
}
```

Рисунок 3.6 – Приклад передачі даних з FT2232 мовою C++ на OS Linux

Кроки для підключення FT2232 до плати розробки (Arduino Nano) за протоколом UART наступні.

1. Визначити піни UART на Arduino Nano. Arduino Nano має піни TX та RX, які використовуються для UART-комунікації. Також важливо визначити GND (заземлення), оскільки це обов'язково для створення електричної взаємодії між пристроями. Треба зробити такі підключення :

– підключення FT2232 до Arduino Nano;

– підключити TXD FT2232 до піна RX Arduino Nano;

– підключити RXD FT2232 до піна TX Arduino Nano;

– під'єднати GND FT2232 з GND Arduino Nano.

2. Живлення FT2232. Підключити VCC FT2232 до відповідного джерела напруги (зазвичай від 1.8V до 5.25V). Дана плата може живитися від

USB-порту комп’ютера. Для цього важливо визначити необхідний струм для реалізації схеми, проєкту тощо.

3. Перевірка налаштувань. Перевірити налаштування UART для Arduino Nano та FT2232. Встановити швидкість передачі даних (бод) та порт підключення до комп’ютера (може налаштовуватися автоматично на порт COM3 для Windows або /dev/ttyUSB0 для Linux).

4. Підключення до комп'ютера. Підключити FT2232 до USB-порту комп'ютера, для зв'язку з Arduino Nano.

Після цих кроків FT2232 буде готовий взаємодіяти з Arduino Nano через UART. Варто зазначити, що також може бути необхідно встановити відповідні драйвери для FT2232 на комп'ютері, якщо вони ще не встановлені.

Також дані кроки універсальні для інших плат розробки. В даній роботі одна з таких плат - це STM32F103C8T6 (Bluepill).

```
void setup() {
  // Ініціалізація Serial зі швидкістю 9600 бод
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    // Читання даних з UART
    String receivedData = Serial.readStringUntil('\n');

    // Виведення отриманих даних
    Serial.print("Received: ");
    Serial.println(receivedData);

    // Відправлення відповіді
    Serial.println("Hello, Python!");
  }
}
```

Рисунок 3.7 – Мінімально необхідний код Arduino (Nano, Uno тощо) для імплементації зв’язку з FT2232 за допомогою UART

3.3 Проведення діагностичного експерименту

Сутність ДЕ з різними протоколами зв'язку в МК системах імітаційного моделювання полягає у програмній імплементації передачі трьох бітів керуючої послідовності з FT2232 до Arduino Nano через UART та до STM32F103C8T6 через SPI. Мова програмування для FT2232 - Python. Реалізація виконана на Linux OS. Мова програмування Arduino та STM - C. Вибір саме цих протоколів обумовлен можливістю їх локальної реалізації без використання обладнання комп'ютерних мереж. Тестер у вигляді ПК імітує часові параметри роботи пристрою керування.

Перед реалізацією необхідно інсталювати дві бібліотеки pyserial та spidev. В терміналі операційної системи ввести дві команди pip3 install pyserial та pip3 install spidev. На рис. 3.8 зображено повний код формування та передачі необхідних даних до плат розробки. Варто зауважити, що змінні порту та бодрейту, шини та девайсу встановлені за замовчуванням та можуть бути змінені, згідно з автоматичними налаштуваннями операційної системи після підключення пристроїв до живлення та FT2232. Дані посилаються циклічно та з невеликою затримкою для того, щоб впевнитися, що вони дійшли успішно та без пошкоджень пакетів.

```
import serial
import spidev
import time

def uart_send(data, port='/dev/ttyUSB0', baudrate=9600): #set the default port to ls /dev/tty/ command in Linux and set the standard baudrate
    # setting the UART
    ser = serial.Serial(port, baudrate)
    time.sleep(2)  # UART initialization delay

    # Send the data
    ser.write(bytes([data]))

    # close UART connection
    ser.close()

def spi_send(data, bus=0, device=0): #set the default SPI bus and device in Linux
    # setting the SPI
    spi = spidev.SpiDev()
    spi.open(bus, device)

    # Відправка 3-бітного числа через SPI
    spi.xfer([data])

    # Закриття SPI
    spi.close()

if __name__ == "__main__":
    # send UART data
    uart_data = 3
    uart_send(uart_data)

    # send SPI data
    spi_data = 3
    spi_send(spi_data)
```

Рисунок 3.8 – Код для встановлення звязку FT2232 з Ardino Nano та STM32 за допомогою UART та SPI на мові програмування Python та Linux OS

Заголовочний файл програмної реалізації двох плат розробки відрізняється мінімально. В даному файлі встановлені макроси затримок згідно з графом переходів пристрою, що розробляється, перелік станів та порти виведення інформації для Arduino (рис.3.8) та STM (рис.3.9) відповідно, що являється головною відмінністю. Також апаратний таймер Arduino має мікросекундний діапазон вимірювання часу, тому створений додатковий макрос конвертації затримки у мілісекундах в мікросекунди.

```
#ifndef __MAIN_H
#define __MAIN_H

#define to1 1000 //timeout for a1 and a7
#define to2 2000 //timeout for a2 and a4
#define to3 5000 //timeout for a3 and a5
#define to6 4000 //timeout for a6
#define toi 10   //immediate timeout for GW pushing analyzing

#define MS_TO_MCS 1000 //Timer in microseconds need to multiply the timeouts

#define R   D2
#define G   D3
#define YRG D4
#define YGR D5

typedef enum {a1, a2, a3, a4, a5, a6, a7} states; //enum of states

#endif /* __MAIN_H */
```

Рисунок 3.9 – Файл-заголовок для Arduino Nano

```
#define TRAFFICLIGHT_GPIO GPIOA //define the input GPIO port

//Outputs for STM32
//
#define R   GPIO_PIN_5
#define G   GPIO_PIN_6
#define YRG GPIO_PIN_7
#define YGR     GPIO_PIN_8
```

Рисунок 3.10 – Відмінність файлу-заголовку STM32F103C8T6

Необхідні змінні створені та проініціалізовані за замовчуванням. ключове слово volatile вказує на змінні, які використовуються у перериваннях за таймером та передачі даних (тільки SPI) для того, щоб не видозмінити дані при опрацюванні їх в різних слоях реалізації.

Використовуючи інтерфейс передачі даних UART на платі Arduino Nano, слід перевірити чи дійсно даний спосіб фізично працює та піни не знаходяться “у повітрі”. Після даної перевірки, читається до кінця пакет та необхідні дані заносяться у відповідні змінні. Дані змінні використовуються для умов функції переходів (рис.3.11).

```
//whether the UART is enabled
if (Serial.available() > 0) {
  //collect data and parse as number
  unsigned char receivedNumber = Serial.parseInt();

  //when the message has been read, then collect each bit as the input values
  if (Serial.read() == '\n') {
    St  = (receivedNumber>>0)&0x01;
    Onn = (receivedNumber>>1)&0x01;
    GW  = (receivedNumber>>2)&0x01;

  }
```

Рисунок 3.11 – Отримання даних з UART

Функція переходів – це програмна реалізація переходів автомата із використанням шаблону switch–case. Кожен case описує певний стан, в якому зараз знаходиться автомат та подальший перехід у наступний стан при певних логічних умовах. На лістингу 3.1 зображено фрагмент реалізації функції переходів, а саме можливість переходу у режим “Зелена хвиля”. В разі позитивного сигналу на GW, який був переданий через UART, вираховується час, за який була, умовно, натиснута кнопка для виклику даного режиму. Якщо сума затримки режиму GW більше за затримку стану a3, режим виконується майже миттєво, оновивши дані часу таймера. При закінченні даного режиму, підраховується кількість часу, який лишився до повного закінчення перебування автомату а стані a3 та за логікою або повертається у стан a3, допрацьовуючи його. Або зі стану a6 одразу після закінчення переходить у стан a4.

Функція затримок демонструє логіку переходу з поточного стану у наступний, вираховуючи необхідну затримку та встановлюючи новий період переповнення таймеру. Надалі автомат працює у штатному режимі

Лістинг3.1– Функція переходів і затримок у режимі ”зелена хвиля ”

```
   case a3:
    if(St&Onn&!GW) nextState=a4;
    else if(St&Onn&GW)
       //count how much time left from to3 and compare sum with to6 it with standard to3
       //once it will not smaller than to3, so the GW is enabled almost immediately
              diffto=to3-(millis()%to3);
              if(diffto+to6>=to3){ nextState=a6;
            Timer1.setPeriod(toi * MS_TO_MCS);  }
    else if (!St|!Onn) nextState=a1;
   break;
...
   case a6:
    if(Re) nextState=a4;
    else nextState=a3;
   break;
...
      // Delay function, calculation Re
   case a6:
    if(to6+diffto<to3)  Re=0;
    else  Re=1;
     timeout = to6;
   break;
```

Програмна реалізація для STM32F103C8T6 ненабагато відрізняється синтаксисом. Логіка роботи функції переходів, виходів та затримок лишилася незмінною.

Отримання даних через SPI відбувається методом обробки переривання. Даний метод є найшвидшим та найструктурнішим. Як і в Arduino в головному циклі програми викликається переривання. Після обробки переривання, отримані дані записуються у змінні обробки оповіщувальних сигналів та подій (рис. 3.12)

Перед тим встановлюється переривання за переповненням таймеру. Початкова затримка встановлюється вручну та може бути змінена у параметрах проєкта.

```
/* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3); //Allow to start the transition function

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  //using the SPI receiving with interrupt mechanism
  HAL_SPI_Receive_IT(&hspi1, &spiRxBuffer, 1);

     //when the message has been read, then collect each bit as the input values
     St  = (receivedNumber>>0)&0x01;
     Onn = (receivedNumber>>1)&0x01;
     GW  = (receivedNumber>>2)&0x01;
```

Рисунок 3.12 – Отримання даних SPI

Функція затримок – це обробник переривань за переповненням таймера Вона відрізняється синтаксисом виконання системних функцій та команд для перевірки, який таймер переповнився та викликав переривання. Зупинкою обробки переривання, зміною стану, перезапуском таймера з відповідною встановленою затримкою.

Функція виведення даних на STM32F103C8T6 (рис.3.13) відрізняється тільки синтаксисом використання функції, яка займається виведенням даних на піни. Виведення інформації відбувається за поточним станом у головному циклі програми.

```
     /* OUTPUTS FUNCTION START */
       switch(currentState)
{
  case a1:
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,R,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,G,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,YRG,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,YGR,GPIO_PIN_RESET);
  break;

  case a2:
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,R,GPIO_PIN_SET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,G,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,YRG,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(TRAFFICLIGHT_GPIO,YGR,GPIO_PIN_SET);
  break;
```

Рисунок 3.13 – Функція виведення

На рис.3.14 зображено обробник переривань отримання даних через SPI. Даний обробник використовує доступ до регістру, який містить в собі отриману інформацію та заносить всі дані з нього у змінну, яка надалі використовується в головному циклі.

```
void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi)
{
  if (hspi->Instance == SPI1) // Змініть належним чином для вашого SPI інстансу
  {
    // Отримання даних
    spiRxBuffer = hspi->Instance->DR; // Читання регістра даних
  }
}
```

Рисунок 3.14 – Обробник переривання SPI

Діагностичний експеримент бувло проведено за допомогою логічного аналізатора Salea Logic Analyzer. Даний пристрій дозволяє в режимі реального часу визначити стан сигналів, що передаються до портів введення-виведення мікроконтролеру. Також може аналізувати та відображати інформацію передачі даних за інтерфейсами (UART, SPI, 1-Wire тощо).

Діагностичний експеримент проведений із використанням протоколів передачі даних UART та SPI. Дані протоколи використані для повної та необхідної реалізації алгоритму, програмної та апаратної реалізації та доцільності використання пакетів передачі даних у розмірі одного байта без додаткових налаштувань.

В результаті проведення ДЕ по обходу графа по маршруту a1-a2-a3-a4-a5-a2 та виклику зеленої хвилі (a3-a6-a4) отримані майже ідентичні часові діаграми (рис. 3.15 та 3.16), окрім часових проміжків, коли були відправлені перші дані до плат розробки та занесені до сигналів оповіщення (St, Onn) та події (GW). Отринані часові діаграми (waveforms - WFs) в свою чергу співпадають зі специфікацією (рис. 3.3).

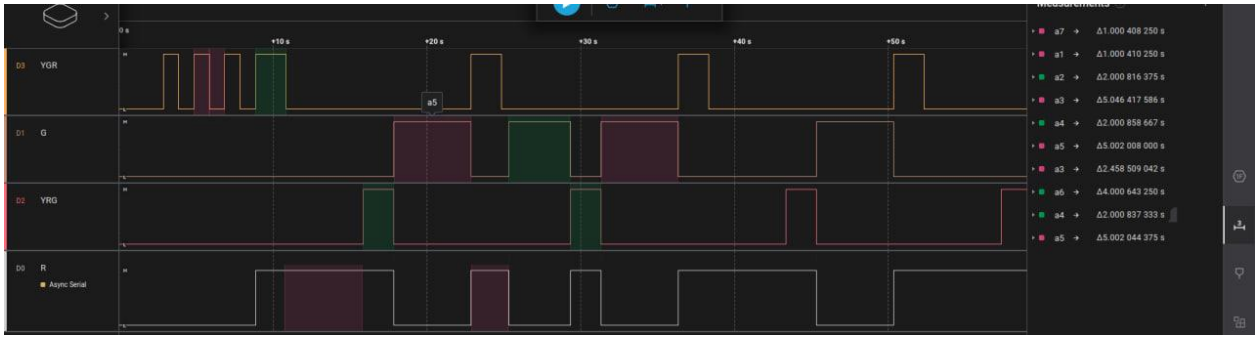

Рисунок 3.15 - Waveform для протоколу UART та плати Arduino Nano

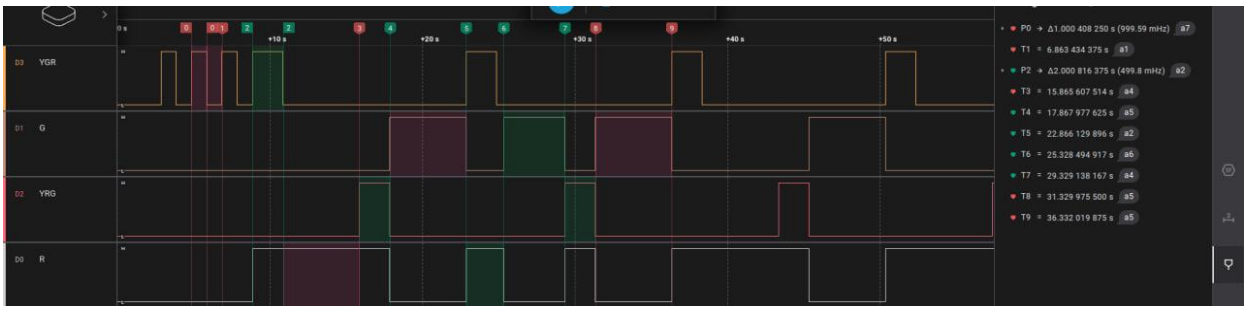

Рисунок 3.16 - Waveform для протоколу SPI та плати STM32F103C8T6

Часові діаграми мають чотири сигнали YGR, G, YRG та R, які є пінами виведення даних, тобто вихідними сигналами. Завдяки зміні значень даних сигналів, можна побудувати структуру переходів автоматного шаблону.

Часова діаграма UART або SPI має позначені двома кольорами проміжки часу, в яких відбувається зміни сигналів відповідно до стану автомата і за певною затримкою. Затримки відображені праворуч від самої діаграми. Оскільки виділення проміжків часу відбуваються вручну, то похибка виміру часу, скільки сигнал дорівнює “0” або “1” може бути неточним. Похибка є настільки мінімальною, що нею можна знехтувати.

Парні стани на вейвформах позначені зеленим кольором, а непарні - червоним.

Робота пристрою розпочинається з передачі байту зі значенням “0”. При даній передачі даних, робота автоматного шаблону не виконується що імітує відключений пристрій. Згідно з усіма сигналами вейвформи на

початку, це стан a1.

При зміні значення сигналу Onn на “1” розпочинається “аварійний” режим роботи пристрою, що ідентифікується періодичною зміною сигналу YGR. Це відображає роботу станів a7-a1.

При позитивному значенні сигналу St (байт даних UART/SPI містить два останніх біти зі значенням одиниці - 00000011(2) - 3(10)) відбувається перехід у “штатний” режим роботи. Виконується послідовність переходів a1-a2-a3-a4-a5-a2 до моменту, поки один із сигналів оповіщення не стане “0” або не виконується умова “Зеленої хвилі”.

Розглянемо окремо роботу виклику “Зеленої хвилі”. Подія GW є одним з бітів при передачі даних (байт даних UART/SPI містить три останніх біти зі значенням одиниці - 00000111(2) - 7(10)). За імплементованою програмною логікою, Дана подія має прийти не пізніше, ніж за 4 секунди після початку очікування, перебуваючи у стані a3. В даних вейвформах виконується умова, що “Зелена хвиля” вмикається майже після 2,5 секунд очікування. Умова виконується і автомат миттєво переходить у стан a6 і стабільно пропрацьовує 4 секунди, відображаючи “позитивний” сигнал R.

Після закінчення режиму “Зеленої хвилі”, за програмною реалізацією, якщо лишається час від затримки у стані a3 після роботи стану a6, автомат має повернутися у стан a3 і допрацювати залишившийся час. Однак у даному випадку з 5 секунд, відведених на стан a3, 2.5 секунди було витрачено на виклик режиму “Зелена хвиля”. Даний режим тривав 4 секунди. Оскільки 2.5 секунди менше за повний час роботи стану a6, відбувається перехід у стан a4, який відображений у вейвформах сигналами YRG та R.

Далі зі стану a4 відбувається стандартний перехід у стан a5 і автоматний шаблон вертається у “штатний” режим роботи.

Аналогічний режим роботи відображений і на SPI waveform, що доводить про ідентичність передачі даних, в незалежності від протоколу, який було обрано для імплементації.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи розглянуті питання налагодження та діагностування мікроконтолерних систем логічного управління в реальному часі.

Проведений аналіз методів налагодження мікроконтролерних систем логічного управління в реальному часі та показана неможливість їх налагодження в покроковому режимі.

Показано, що при діагностиці систем логічного управління реального часу доцільно використовувати імітаційне налагодження при якому тестер імітує не тільки логічні умови алгоритму функціонування, а і оповіщувальні сигнали об'єкта управління в реальному часі. При цьому виділено три рівні імітації: інтерфейсний, процесорний і програмний рівні.

При тестовому діагностуванні мікроконтролерних систем вважається, що сам МК (тобто апаратна частина системи) не має дефектів за визначенням. Тестування програмної частини здійснюється, як правило, позитивними тестами замовника на рівні верифікації за правилами тестування програмного забезпечення. Таким чином найбільш важливою частиною при налагодженні мікроконтролерної системи взаємодії з периферією є інтерфейсний рівень налагодження.

В якості об'єктів діагностування розглянуті мікроконтролерні налагоджувальні плати Arduino, STM та ESP. В якості інтерфейсів зв'язку розглянуті протоколи UART (Universal Asynchronous Receiver/Transmitter), USART (Universal Synchronous/Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I^2C (Inter-Integrated Circuit), CAN (Controller Area Network), та LIN (Local Interconnect Network), які мають свої унікальні характеристики та галузі застосування.

В якості тестера запропоновано використовувати одноплатний комп'ютер, а для уніфікації протоколів та зв'язку з об'єктами діагностування

використовувати інтегрований перетворювач інтерфейсів FT2232.

Для перевірки запропонованого метода налагодження мікроконтролерних моделей часових автоматів розглянуто систему управління дорожнім світлофором з реалізацією функції «зелена хвиля» (Gw) для автомобілів спеціальних служб, а саме керуючий автомат цього пристрою, модель якого написана на мові програмування С для мікроконтролеров Arduino та STM.

Діагностичний експеримент було проведено за допомогою персонального комп’ютера в якості тестера, налагоджувальних плат Arduino та STM, протоколів передачі даних UART та SPI і логічного аналізатора Salea Logic Analyzer. Отримані часові діаграми співпали зі специфікацією пристрою керування.

Наукова новизна роботи полягає у подальшому розвитку методів імітаційного тестування мікроконтролерних систем реального часу на інтерфейсному рівні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Shkil A. Design of real-time logic control system on FPGA / M. Miroschnyk, A. Shkil, E. Kulak, D. Rakhlis, I. Filippenko, M. Hoha, M. Malakhov, V. Serhiienko // Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS'19), Sept. 13-16, Batumi, Georgia, 2019. – P.488-491.

2. Способы отладки встраиваемых микропроцессорных систем в преобразовательной технике [Електронний ресурс] / НПФ ВЕКТОР. – URL: https://motorcontrol.ru/production/soft/vector_debug_software – Дата звернення: 10.09.2023.

3. Зайцев В. Г. Комп’ютерні системи реального часу: навчальний посібник / В.Г. Зайцев, Є.І. Цибаєв. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 162 с.

4. Луканов, А. С. Системы реального времени: учебное пособие / А.С. Луканов. – Самара: Издат. Самарского университета, 2020. – 156 с.

5. Мікропроцесори та мікроконтролери: Курс лекцій КПІ ім. Ігоря Сікорського; уклад.: Д.Д. Татарчук, Ю.В. Діденко. – Київ: КПІ ім. Ігоря Сікорського, 2020. – 238 с.

6. Pankov D.A., Automated testing and fault diagnosis of the microcontroller system / D.A Pankov, L.A Denisova // 2019 | IOP Conference Series: Materials Science and Engineering, 2019, Volume 537, part 022072. – 7 p. [Electronic resource] Institute of Physics, London, UK – Mode of access: https://fatcat.wiki/container/ze5oxgkzabgozpk5uwjjucu7sa/browse?volume=537 – Date of access: 10.10.2023. doi:10.1088/1757-899x/537/2/022072.

7. Камкин А.С. Верификация микропроцессоров: борьба с ошибками и управление качеством [Електронний ресурс] / Электроника: НТБ. – 2010. – № 3. – С. 98-104. – Режим доступу: URL: http://www.electronics.ru/journal/article/57 – Дата звернення: 10.02.2023.

8. Слинкин Д.И., Анализ современных методов тестирования и

верификации проектов сверхбольших интегральных схем / Д.И. Слинкин // Программные продукты и системы (Software & Systems). – 2017. – Т.30. – № 3. – С. 401-408.

9. Serial Communication Protocols and Standards RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX / D.S. Dawoud, P. Dawoud. – River Publishers, 2020 – 532 p. [Електронний ресурс] / IEEE Xplore Digital Library – Режим доступу URL: https://ieeexplore.ieee.org/book/9218891. – Дата звернення: 15.11.2022.

10. Future Technology Devices International Ltd FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC [Electronic resource] – FTDI Chip. – Mode of access: https://www.ftdichip.com/old2020/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf – Date of access: 10.09.2023.