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)

Технологія тестування цифрових пристроїв виробником за сценаріями замовника
(тема)

Виконав: студент 2 курсу, групи СКСм-20-1

_____Федосов Д.Ю._____
(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 проф. Свірь І.Б.__
(посада, прізвище, ініціали)

Допускається до захисту

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

2022 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)

«__07__»__11____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Федосову Данілу Юрійовичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Технологія тестування цифрових пристроїв виробником за сценаріями замовника_____
Digital Devices Testing Technology by the Designer According to Customer Scenarios _____

затверджена наказом по університету від «__14_»__11__2022 р. № __1478 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____20.12.2022_____

3. Вихідні дані до роботи (проекту)_____
Мікроконтролер STM32_____
Плата розробки Arduino_____
Мова програмування Сі_____
Середовище розробки Keil for ARM _____
_____

4. Перелік питань, що потрібно опрацювати у роботі _____
Принципи взаємодії замовника та розробника програмного забезпечення____
Технології тестування вбудованих систем_____
Технології діагностування мікроконтролерних пристроїв_____
Особливості тестування мікроконтролерних програм_____

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 02.09.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2022-08.09.2022	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2022-15.09. 2022	
3	Аналіз технологій діагностування мікроконтролерних систем	16.09.2022-29.09.2022	
4	Аналіз взаємодії замовника та розробника програмного забезпечення	30.09. 2022-13.10.2022	
5	Розробка технології взаємодії замовника і потужностей виробництва	14.10. 2022-31.10. 2022	
6	Проведення діагностичного експерименту для мікроконтролерного пристрою	01.11. 2022-15.11. 2022	
7	Формування тестового звіту по результатам тестування	15.11. 2022-30.11. 2022	
8	Оформлення пояснювальної записки	01.12. 2022-15.12. 2022	
9	Захист проекту	15.12. 2022-25.12. 2022	

Студент _____
(підпис)

Керівник роботи (проекту) _____ проф. Свірь І.Б.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 63 сторінку, 33 рисунки, 9 джерел за переліком посилань.

МІКРОКОНТРОЛЕР, ARDUINO, STM. ТЕСТУВАННЯ, ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС, ЛОГІЧНИЙ АНАЛІЗАТОР, ТЕСТОВИЙ ЗВІТ

Метою проєкту є імплементація методу тестування цифрових мікроконтролерних пристроїв потужностями виробництва за сценаріями замовника, узгодженими з обох сторін.

Апаратна реалізація методу тестування мікроконтрлолерних пристроїв на різних технологічних платформах складається з двох плат, які узгоджено правилами підключення без відхилення від коректності роботи. Платою для тестування (ПДТ) обрана плата на базі мікроконтролера STM32F103C8T6. Платою, що тестується (ТП) обрана плата Arduino UNO. Програмна реалізація ПДТ виконана за допомогою середовища CubeMX та Keil for ARM. Програмна реалізація ТП виконана в середовищі Arduino IDE.

Результат програмно-апаратної реалізації відображаються за допомогою зчитування сигналів логічним аналізатором Saleae Logic від ТП. Моделювання роботи системи проведено за «позитивним» та «помилковим» тестовим сценарієм. Результати тестування надсилаються від Виробника до Замовника у вигляді класичного тест-звіту з переліком кількості проведених тест сценаріїв та дефектів, що діагностовано.

ABSTRACT

The explanatory note contains: 63 pages, 33 figure, 9 sources according to the list of links.

MICROCONTROLLER, ARDUINO, STM. TESTING, SOFTWARE AND HARDWARE COMPLEX, LOGIC ANALYZER, TEST REPORT

The goal of the project is the implementation of the method of testing digital microcontroller devices by production capacities according to customer scenarios agreed on by both sides.

The hardware implementation of the method of testing microcontroller devices on different technological platforms consists of two boards, which are agreed upon by the rules of connection without deviating from the correctness of operation. A board based on the STM32F103C8T6 microcontroller was selected as the board for testing (BFT). An Arduino UNO board is selected as the board under test (TB). The software implementation of BFT is made using the CubeMX environment and Keil for ARM. The software implementation of TB is made in the Arduino IDE environment.

The result of software and hardware implementation is displayed by reading signals with a Saleae Logic logic analyzer from TB. Simulation of the system operation was carried out according to the "positive" and "false" test scenarios. The test results are sent from the Manufacturer to the Customer in the form of a classic test report with a list of the number of tested scenarios and diagnosed faults.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГСА – граф-схеми алгоритмів;

ДП – дискретний пристрій;

КА – керуючий автомат;

МК – мікроконтролер;

ОА – операційний автомат;

СЛУ – системи логічного управління;

ПДТ – плата для тестування;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

ПЛІС – програмовані логічні інтегральні схеми;

САПР – система автоматизованого проектування;

ТП – плата, що тестується;

ТС – тестова специфікація;

ЦП – цифровий пристрій;

FSM – finite state machine (кінцевий автомат);

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

HDL – hardware description language (мова опису апаратури);

RTL – register transfer level (рівень регістрових передач);

SoC – System of Chip (система на кристалі);

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

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

ВСТУП

Розробка програмного забезпечення не є безперервним процесом, а складається з кількох обов'язкових кроків. Для того щоб організувати робочий процес з максимальною вигодою для всіх, необхідний набір ефективних інструментів і практик. Ось чому, вибираючи команду розробників, потрібно орієнтуватися не тільки на їхній досвід та кваліфікацію, але й на кращі практики управління розробкою програмного забезпечення, які вони використовують.

Ринок розробки програмного забезпечення пропонує великий вибір інструментів управління розробкою ПЗ. Кожна з методологій, які використовуються в процесі розробки програмного забезпечення, має свої особливості використання. Тому при виборі конкретної практики потрібно враховувати всі можливі фактори, які можуть вплинути на процес та результат роботи.

Представники замовників є найважливішою ланкою успішної розробки програмного забезпечення. Представники мають не просто контактувати, а буквально фізично бути присутнім у безпосередній близькості та працювати в команді розробників. Будь-яка проблема повинна бути виявлена на ранньому етапі, будь-які побажання або питання повинні вирішуватись у реальному часі.

Представники замовника є джерелом історій користувачів та тестових наборів даних, вони беруть участь у плануванні плану релізів. Крім того, відкритий процес дозволяє інспектувати спірні ділянки коду будь-який момент часу, створюючи повністю прозору атмосферу між розробниками та замовниками. Хоча це не очевидно, але практично відбувається економія часу замовника, який весь час перебуває в курсі справ – додатково час економиться на детальних специфікаціях на початку роботи, оскільки будь-який аспект можна з'ясувати у процесі роботи. Згодом не доведеться

інструктувати персонал замовника, оскільки замовник уже має в своєму розпорядженні високоякісні фахівці з даного продукту.

При цьому багато документів-посередників стають непотрібними, оскільки багато що вирішується усно, без залучення технічних та бюрократичних механізмів. Значення мають тільки кінцеві результати роботи – але не проміжні рішення та дискусії, тож ні необхідності документувати всі можливі ходи думки та альтернативи.

Таким чином, актуальною задачею є розробка моделі взаємодії розробника (проектувальника) з замовником, особливо на етапі передачі замовнику готового виробу та технічної документації.

1 ТЕСТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ У ВИРОБНИЦТВІ

1.1 Взаємодія замовника та розробника програмного забезпечення

Практично будь-який електронний пристрій, це логічний пристрій, що складається з мікроконтролерів та схем сполучення з периферійними пристроями (іншими електронними пристроями, датчиками, виконавчими механізмами). У більшості мікроконтролерів є пам'ять програм, яка є деякою кількістю осередків (від тисячі до десятків тисяч). У цих осередках зберігаються та сама програма (програмне забезпечення), яку виконує мікроконтролер.

Програма (програмне забезпечення, ПЗ) – це послідовність команд, що виконуються мікроконтролером. Кожній команді у пам'яті відповідає своє число – код. При включенні живлення мікроконтролер один за одним зчитує ці коди, здійснює їхню дешифрацію (визначає, що йому потрібно зробити), а потім виконує одну за іншою ці дешифровані команди. Головна особливість пам'яті, це те, що занесені до неї коди, зберігаються незмінними, за відсутності живлення мікроконтролера. Саме програмне забезпечення визначає поведінку всієї системи. Всі зміни та модифікації програмного забезпечення вносяться до мікроконтролера за допомогою програматора. Один і той же мікроконтролер, але з програмним забезпеченням, написаним під певне завдання, може застосовуватися в різних системах – для керування котлом теплопостачання або пристроїв автоматизованого телефонного зв'язку. У процесі створення програмного забезпечення програміст пише текст програми на комп'ютері так само, як він пише будь-який інший текст. Потім він запускає спеціальну програму транслятор. Транслятор – це спеціальна програма, яка переводить текст, написаний програмістом, у машинні коди, тобто у форму, зрозумілу мікроконтролера. Написаний програмістом текст програми називається вихідним кодом. Код, отриманий в

результаті трансляції, називається результуючим чи машинним кодом. Саме цей код записується у програмну пам'ять мікроконтролера за допомогою програматора

Від того, як написано програмне забезпечення залежить, як працюватиме сам пристрій, його швидкодія, завантаження. Швидкість відпрацювання мікроконтролером програмного забезпечення, написаного програмістами різного рівня, може відрізнятися у рази. Хоча розробка мікроконтролерного ПЗ має деяку специфіку, вона в цілому відповідає процедурам розробки загального програмного забезпечення.

Розробка програмного забезпечення в більшості випадків повинна розглядатися як колективна праця фахівців, спрямована на задоволення потреб користувачів автоматизації їх діяльності. Це процес, часом тривалий, що пов'язує виробничими та іншими відносинами тих, кого тією чи іншою мірою можна розглядати як виробників програми. Як і будь-яка праця, тісно пов'язана з неоднозначними потребами тих, хто використовуватиме продукти праці, необхідним елементом розробки програм є вирішення завдань вивчення користувачів, з одного боку, а з іншого – забезпечення зворотного зв'язку з ними, що спрямовує виробництво. Це складові, у тому числі формуються основні завдання управління виробництвом програм. Найчастіше вирішення таких завдань здійснюється керівником, або, як заведено говорити, менеджером проекту.

Поняття «менеджер проєкту» необов'язково співвідноситься з конкретною персоною, яка відповідає за керування виробництвом програмної системи загалом. У невеликому проєкті така роль найчастіше виправдана: вона дозволяє концентрувати всі нитки управління, виключає проблеми внутрішнього для проєкту узгодження протиріч, забезпечує централізовану відповідальність за проект перед тими, хто зацікавлений у його виконанні. Або менеджерські якості успішно бере на себе один з командних гравців (розробник, тімлід, найстарший за рівнем, тощо) і автоматично стає учасником та посередником між власною командою і замовником. Нерідко

невеликі проєкти хоч і мають достатньо великий бюджет, контролюються виключно замовником (рис.1.1).

Однак у міру переходу до масштабніших проектів менеджерські обов'язки стає неможливо концентрувати в одних руках. Зазвичай у разі надходять відповідно до схеми організації виробництва з утворенням групи менеджерів, відповідальних за різні розмежовані сфери проекту.

У цій схемі централізація досягається шляхом призначення головного менеджера проєкту, який делегує повноваження менеджерам за напрямами (рис.1.1).

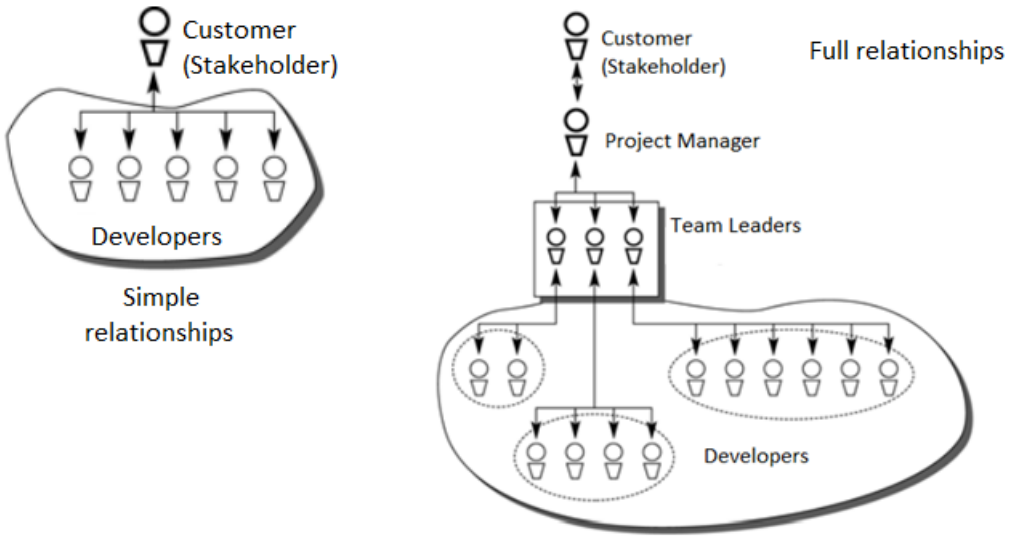


Рисунок 1.1 – «Класична» модель відносин замовника та розробника

У колективах, які виконують програмні проекти, можливі найрізноманітніші організаційні структури. Так, для складних, об'ємних за трудовитратами проектів ієрархічні ланцюжки «менеджер – менеджер за напрямом – виконавець робіт» доцільно збільшувати, диференціюючи роботи та сфери відповідальності для деяких менеджерів за напрямом.

Іноді виправдане делегування всіх менеджерських обов'язків та повноважень менеджерам за напрямами. В результаті відповідальність за проект несе не одна людина, вона розподіляється по менеджерській групі. Таким чином з'являється колективна, деперсоніфікована відповідальність.

Можливі схеми, коли замість менеджерів за напрямами деперсоніфікована відповідальність призначається групі загалом. Менеджер проєкту видає завдання, контролює їх виконання, здійснює інші функції, звертаючись до всієї групи, всередині якої вже розподіляються обов'язки, у тому числі обов'язки менеджера за напрямом.

Для невеликих груп можлива наступна організація робіт: обов'язки головного менеджера розподіляються за командою розробників, яка за рахунок внутрішніх механізмів вирішує, як планувати роботи, як їх розподіляти та контролювати. Це для так званого підходу швидкої розробки (agile development) програмних систем, що об'єднав у собі кілька методологій програмування, які відмовляються від багатьох принципів традиційного менеджменту. «Agile» – це комплексна методологія. Ітеративний підхід до управління проектами та розробки програмного забезпечення, який допомагає командам швидше та з меншими проблемами виконувати вимоги клієнтам. Замість того, щоб випустити весь продукт повністю, команда, яка дотримується принципів «Agile», виконує роботу в рамках невеликих, але зручних етапів-інкрементів. Вимоги, плани та результати оцінюються безперервно, тому команди можуть швидко реагувати на зміни (рис.1.2). Одним з практичних методів використання «Agile» є техніка «Scrum».

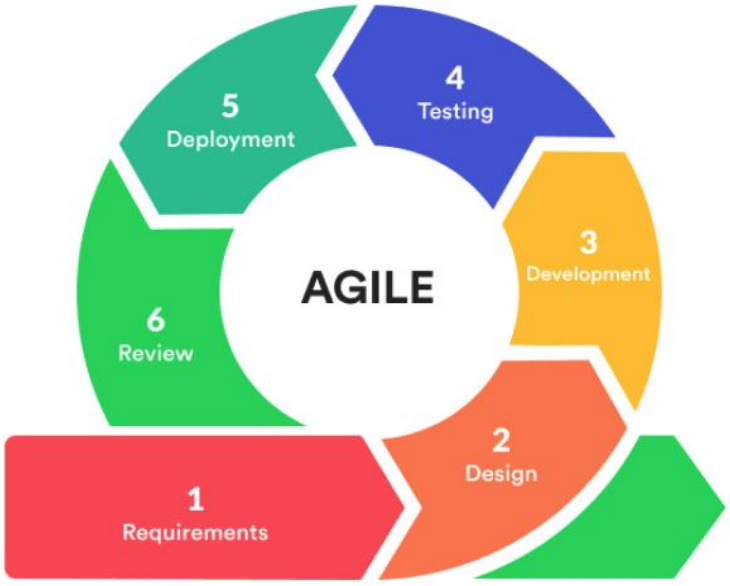


Рисунок 1.2 – Етапи розробки продукту за методологією Agile

Основною одиницею документування вимог в Agile-процесах є користувальницькі історії (user stories) форма специфікації вимог, що описує інформацію про очікування і наміри замовника від системи, що розробляється в стилі, що фокусується на мотиваційної складової вимоги і бізнес-цінності для замовника замість конкретного способу його виконання. Користувальницькі історії, як форма вимог, абсолютно не схожі на традиційні специфікації IEEE 830, прийняті у класичних процесах розробки

У «Scrum» (скрам) робиться акцент на планомірному контролі процесу розробки. Головна особливість скраму - це розбивка процесу розробки на ітерації з чіткими відрізками години (зазвичай 2-6 тижнів; їх називають «спринтами» («Sprint»)). На початку спринту проводиться «планування спринту» («Sprint Planning») — нарада на 3-4 години, де обговорюються головні завдання, які виконуватимуться протягом спринту. Наприкінці спринту проводитиметься демо - демонстрація результатів роботи команд за цей спринт (рис.1.3.).

Рисунок 1.3 – Етапи технології «Scrum»

Перед самим першим спринтом замовник, або його представник формує список вимог до майбутнього продукту, який буде розроблятися. Такі вимоги називають «User Story», а самого замовника «Product Owner». У «User Story» «Product Owner» вказує пріоритет за кожним завданням. Спочатку будуть реалізовуватися завдання з більш високим пріоритетом.

Весь список називається «Product Backlog» або «резерв продукту». Окрім «Product Owner», серед учасників ще особливо виділяють «Scrum Master» (скрам майстер). Скрам майстер слідкує, щоб слідували всім принципам скраму. В основному скрам майстер – це або менеджер проєкту («Project Manager») або лідер команди розробки («Tech or Team Lead»).

На першій зустрічі, на початку спринту, крім того, що завданню встановлюється пріоритет, надається ще приблизна оцінка в «абстрактних людських днях», яку ще називають «Story Point». Потім команда вирішує, скільки вона успішно виконає завдань за годину спринту. Наприклад, замовник відображає 7 завдань, а команда зможе зробити лише 5. Значить 2 зайві задачі підуть у наступний спринт. Якщо замовнику така обставина не подобається, він може підвищити їх пріоритет, тоді інші завдання будуть вилучені зі спринту, а ці 2 завдання займуть їх місце. Крім того, скрам майстер може розбити деякі завдання на дрібніші, надати їм пріоритети на свій розсуд, щоб замовник залишився задоволеним.

Список відібраних завдань на спринт називається «Sprint Backlog», або «резерв спринту». Для кращої наочності звичайно складається таблиця, в якій перелічуються завдання, які необхідно реалізувати, які перебувають у процесі реалізації, і ті, що вже виконані (схоже на «канбан» («Kanban») дошку, але не зовсім). Також будується діаграма «згоряння завдань». Вона графічно показує, скільки ще завдань залишилось виконати. В ідеалі графік «згоряння завдань» до кінця спринту має опуститися до нуля.

Також кожного дня або через день проводяться невеликі заходи у командах на 5-15 хвилин, де кожен із учасників перераховує, що зробив за цей день, що ще потрібно зробити, а також вказує на те, що йому щось заважає зробити.

Коли спринт завершився, «Scrum master» проводитиме демо, на якому демонструється список всього, що повністю зроблено. Потім обговорення, яке звичайно намагається виявити, що було зроблено добре, а що можна було зробити краще.

Зазвичай за 2-3 спринти можна виявити головні проблеми, які заважають працювати ефективніше, звісно ці проблеми слід усунути. Такі умови сприяють більшій продуктивності команди, що не збільшує навантаження на команду. До ери гнучких методологій це було неможливе.

Також у середині спринту, іноді ще проводять проміжні зустрічі, присвячені плануванню наступного спринту. На ній звичайно уточнюють пріоритети задач, а також можуть бути якісь завдання поділені, нові завдання додані у «Product Backlog».

Велика кількість як великих, так і малих компаній у світі працюють за даними методологіями та сценаріями. Левова частка таких компаній – це «аутсорінг» та «аутстафінг».

Під «аутсорсингом» («outsourcing») у світовій практиці розуміють передачу певної частини робіт, послуг або бізнес-функцій зовнішньому виконавцю, у якого є необхідні трудові ресурси.

«Аутстафінг» («outstaffing») - це фактична передача персоналу (трудового ресурсу) компанії-виконавця в найм (оренду) компанії-замовнику.

1.2 Відносини та взаємодія замовника і потужностей виробництва

Оскільки «аутсорсинг» та «аутстафінг» займає не 100% ринку розробки, то очевидно, що існують інші види компаній. Одним з таких видів являється «продуктова компанія» (Product Company).

Продуктова компанія займається розробкою власного продукту у вигляді додатку, девайса, сервісу тощо. Продуктом компанії також може бути група будь-яких винаходів, пристроїв, систем тощо. Стартапи також належать до продуктових компаній.

Продукт найчастіше є власністю компанії. І часто для розробки продукту використовується один стек технологій. Але є й винятки.

У великих корпораціях продуктова компанія поєднує в собі й «аутсорсин» проєкти, і продуктову основу, і консалтингові відділи. Так

само й деякі проєктні команди в аутсорсингу роблять продукти на замовлення для клієнта. Оскільки це є вийняток, він не підпадає до розгляду класичної взаємодії продуктової компанії з іншою стороною на ринку розробки і, навіть, виробництва.

Розглянемо детально «продуктову компанію», яка виготовляє пристрої. Існує умова, що ПЗ йде безкоштовно до самого пристрою. За класичною схемою, дана компанія займається розробкою візуального вигляду продукції, її апаратною реалізацією на рівні схем та прототипів та програмною реалізацією від прототипів до реліз-версії. Дана компанія має головний менеджмент (C-Level), який складається з директорів та засновників компанії, менеджерів з продажів, дистрибуції продукції та команди або команд інженерів-розробників, інженерів-тестувальників, інженерів-схемотехніків тощо. В основному, команди об'єднуються під загальну назву «дослідницький центр» (Research and Development – R&D). Ключова робота центру – це розробка пристрою, ПЗ та документації до нього, тестування апаратної реалізації на стадії реалізації схеми та прототипування. Тестування ПЗ від пілотної до релізної версії. Тестування, зазвичай, є комплексними та можуть охоплювати майже всі можливі види. Класична схема взаємовідносин представлена на рис.1.4.

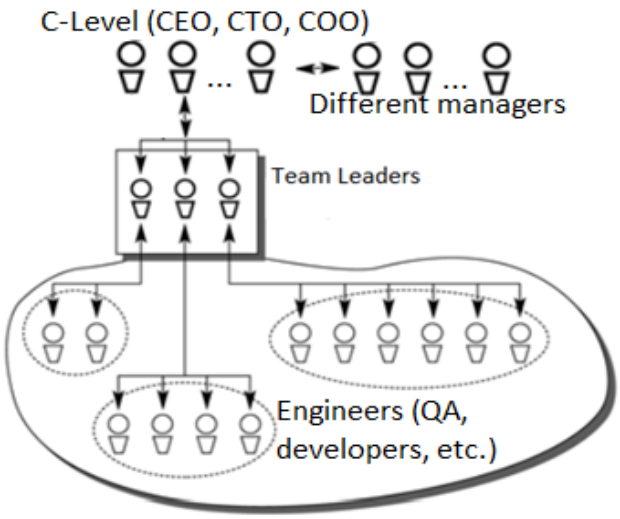


Рисунок 1.4 – Повна класична модель відносин у «продуктовій компанії»

Оскільки у «продуктовій компанії» за класичною моделлю (надалі замовник) відсутня гібридизація, то доцільно мати взаємовідносини із компаніями-виробниками, які будуть вести серійне виробництво продукції, що розробляється. Компанією-виробником (надалі виробник) може бути будь-яка компанія, яка має необхідне обладнання (потужності) для виконання вимог замовника. Замовник і виробник заключають дво- чи тристоронній договір і починають співпрацю.

Відносини замовника та виробника можуть бути узгоджені будь-яким способом. Все залежить від домовленостей у контракті, усних домовленостей, які також можуть бути описані у контракті тощо. Особливо треба приділити увагу відносинам замовника і виробника у процесі виробництва продукції.

У процесі виробництва продукції, учасниками можуть бути як C-Level (рис.1.4, безпосередній замовник), менеджер за відповідним напрямком, тімлід команди чи член команди (розробників, тестувальників тощо), який вповноважений вести діалог з виробником. Розглянемо детально особливості відносин у кожному можливому сценарії.

1. Безпосередній замовник та виробник. В даному сценарії керівництво компанії-замовника та компанії-виробника мають відносини, частина з яких може бути інкапсульована від співробітників за ієрархією. Можуть узгоджуватися абсолютно всі необхідні вимоги з обох сторін (економічні, кадрові тощо). Групи вимог розподіляються за командами, які мають, за своїм напрямом, виконати їх та відзвітувати про результати керівництву, яке в свою чергу передає інформацію виробнику. Взаємодія відбувається за рис.1.5 а).

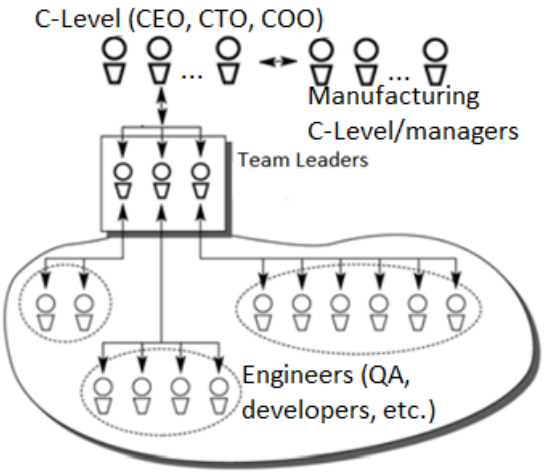
2. Менеджер та виробник. В даному сценарії керівництво отримує дані від менеджеру, який веде напрямок за певним регіоном чи групою або однією компанією-виробником. Можуть узгоджуватися абсолютно всі необхідні вимоги з обох сторін або ж ті, які були попередньо узгоджені з керівництвом. Як і в сценарії з безпосереднім замовником, після узгодження

з керівництвом всіх встановлених відносин, вимоги розподіляються за командами для їх виконання. Частина обов'язків перекладається на менеджера від C-Level. Взаємодія відбувається за рис.1.5 б).

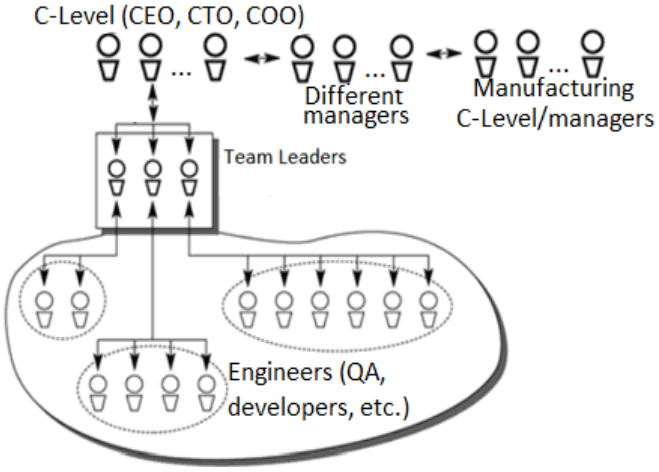
3. Тімлід (лідер команди) та виробник. Даний сценарії має попередньо узгоджений перелік вимог, які може запропонувати лідер команди (розробників, тестувальників тощо) компанії-замовника виробнику. Узгоджені вимоги надаються для перевірки до C-Level та за позитивною відповіддю, делегуються команді для їх виконання. Взаємодія відбувається за рис.1.5 в)

4. Член команди та виробник. Специфічний, але діючий сценарій, в якому виконуються вимоги, надані C-Level, менеджером або лідером команди. У більшості випадків даний сценарій ефективний тим, що член команди виконує обмежений перелік вимог, але взаємодіє з виробником напряму. Існує велика ймовірність пошуку та запровадження змін або нових вимог, за рахунок вмінь та навичок технічного та комунікативного характеру. Взаємодія зображена на рис.1.5 г).

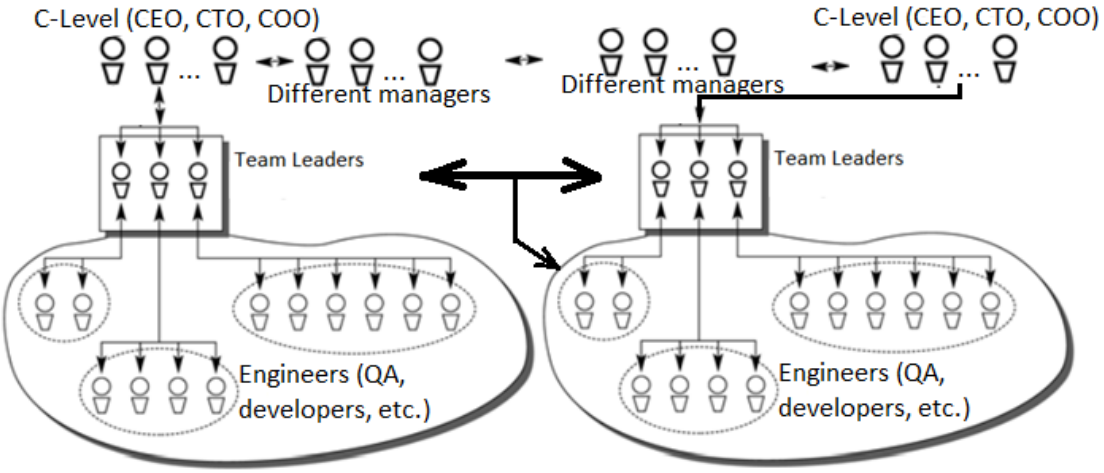
Дані сценарії можна комбінувати та видозмінювати для отримання максимальної ефективності взаємодії з виробником. Варто зазначити, що керівництво компанії-замовника та менеджери спілкується виключно тільки з керівництвом або менеджерами компанії-виробника. Лідери команд з лідерами команд або з членами цих команд. Члени команд компанії-замовника з членами команд компанії-виробника і інколи з лідерами команд. Процеси взаємодії абсолютно симетричні. Результат виконання роботи надається вище за ієрархією штату компанії.



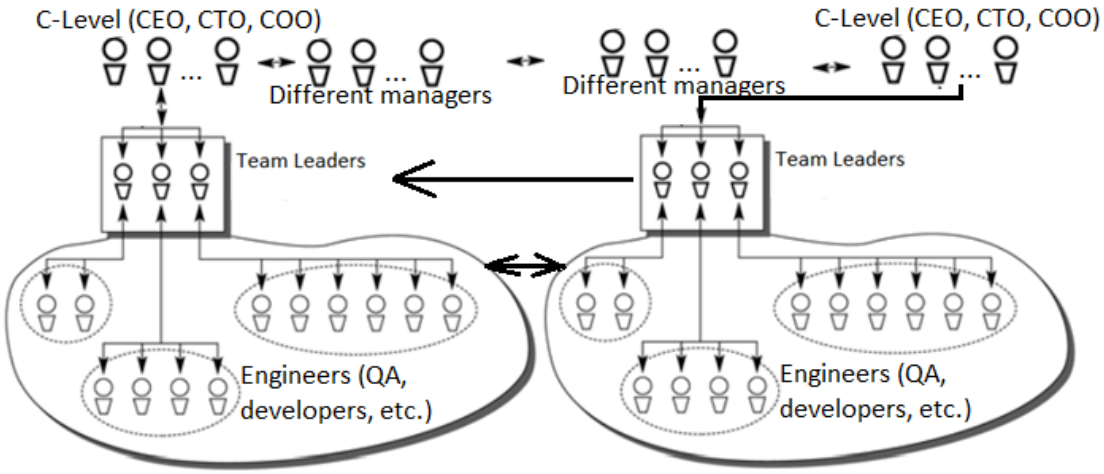
а) Безпосередній замовник та виробник



б) Менеджер та виробник



в) Лідер команди та виробник



г) Член команди та виробник

Рисунок 1.5 – Взаємодія компанії-замовника з компанією-виробником

1.3 Види тестування цифрових пристроїв вбудованих систем

Вбудовані системи – це пристрої з електронним керуванням, у яких програмне та апаратне забезпечення тісно пов'язані між собою. Вбудовані системи можуть містити різноманітні обчислювальні пристрої. Це комп'ютери, інтегровані в інші пристрої для роботи з функціями певної програми. Кінцевий користувач зазвичай навіть не підозрює про їх існування.

Вбудоване тестування – це процес тестування для перевірки функціональних і нефункціональних атрибутів як програмного, так і апаратного забезпечення у вбудованій системі та забезпечення відсутності дефектів у кінцевому продукті. Основною метою вбудованого тестування є перевірка та підтвердження того, чи відповідає кінцевий продукт вбудованого апаратного та програмного забезпечення вимогам клієнта чи ні.

Тестування вбудованого програмного забезпечення перевіряє відповідне програмне забезпечення належної якості та відповідає всім вимогам, яким воно має відповідати. Тестування вбудованого програмного забезпечення є чудовим підходом до гарантування безпеки в таких критичних програмах, як медичне обладнання, залізниця, авіація, автомобільна промисловість тощо. Суворе та ретельне тестування має вирішальне значення для надання сертифікації програмного забезпечення.

Брукман і Нотенбум представляють метод TEmb, який допомагає адаптувати схему тестування для конкретної вбудованої системи. У методі TEmb тестовий підхід для будь-якої вбудованої системи можна розділити на дві частини: загальні елементи, застосовні до будь-якого структурованого підходу, і специфічні для програми заходи. Загальна частина складається, наприклад, із планування тестового проекту, застосування стандартизованих методів, організації команд тестування, спеціального

тестового середовища, звітування тощо. Вони є частинами чотирьох наріжних каменів структурованого тестування: життєвий цикл, інфраструктура, методи та організація, що називаються LITO – lifecycle, infrastructure, techniques and organization. Даний метод можна застосовувати для тестування не всієї вбудованої системи, але й її частини чи взагалі одного пристрою.

За схемою тестування, в основному виділяють, п’ять видів тестування системи, частини, пристрою, а саме:

1. Модульне тестування програмного забезпечення. Модуль одиниці є функцією або класом. Модульне тестування виконується командою розробників, насамперед розробником, і зазвичай виконується за раніше сформованим неофіційним технічним завданням (ТЗ). На основі специфікації модуля розроблено тестові кейси.

2. Інтеграційне тестування. Інтеграційне тестування можна розділити на два сегменти: тестування інтеграції програмного забезпечення та тестування інтеграції програмно-апаратного забезпечення (комплексне інтеграційне тестування). Зрештою перевіряється взаємодія апаратної частини та програмних компонентів. Це може включати дослідження взаємодії між вбудованими периферійними пристроями та програмним забезпеченням. Розробка вбудованого програмного забезпечення має унікальну характеристику, яка зосереджується на фактичному середовищі, у якому працює програмне забезпечення, яке зазвичай створюється паралельно з програмним забезпеченням. Це створює незручності для тестування, оскільки комплексне тестування не може бути виконане в імітованих умовах.

3. Тестування системного блоку. Тепер модуль, який потрібно перевірити, є повною структурою, яка складається з повного програмного коду, а також усіх частин ОСРВ – операційної системи реального часу (RTOS – real-time operating system) і платформи, таких як переривання, механізми завдань, комунікації тощо. Протокол контролю та спостереження

(PCO – Point of Control and Observation) більше не є викликом функції чи методу, а радше повідомленням, надісланим або отриманим за допомогою черг повідомлень ОСРВ. Системні ресурси спостерігаються, щоб оцінити здатність системи підтримувати виконання вбудованої системи. Для цього аспекту тестування сірого ящика є кращим методом тестування. Залежно від організації тестування системного блоку даний вид є обов'язком розробника або спеціальної команди системної інтеграції.

4. Тестування системної інтеграції. Модуль для тестування складається з набору компонентів. Створюється так званий «вузол». Протоколи контролю та спостереження (PCO) – це суміш мережевих протоколів зв'язку та ОСРВ, наприклад мережеві повідомлення та події ОСРВ. Окрім компонента можна використовувати «віртуальний компонент», який також може виконувати роль вузла. Варто зазначити, що даний вид компоненту може бути використаний тільки за відсутності потреби тестувати саме фізичний компонент.

5. Перевірка системи. Модуль, що тестується, – це підсистема з повною реалізацією або повна вбудована система. Метою цього фінального тесту є відповідність функціональним вимогам зовнішнього об'єкта. Варто зазначити, що зовнішня сутність може бути як фізичним, так і інформаційним об'єктом (пристрій, протокол передачі даних, який зв'язує пристрої, підсистему тощо).

Варто зауважити, що дані види тестування завжди мають в собі додаткові методи тестування, такі як «сірий», «чорний» та «білий» ящик, регресійність тощо. Види тестування виконуються в залежності від потреб. Тому не завжди при тестуванні можна зустріти та затвердити всі види. Формування сценарію тестування за всіма видами чи тільки необхідними визначається так званими «викликами» тестування. До них входять: залежність апаратного забезпечення, програмне забезпечення з відкритим кодом дефекти програмного та апаратного забезпечення, постійно чи раптово відтворювані дефекти, необхідні часті або безперервні оновлення

програмного забезпечення.

1.4 Технічне завдання проєкту

Основними проблемами розвитку ринку сучасної мікроелектроніки є зниження вартості і скорочення часу проєктування, що досягається вдосконаленням технічного, інформаційного і програмного забезпечення будь-яких цифрових пристроїв, що розробляються. Особливу увагу слід приділити тестуванню цих пристроїв. Однієї взаємодії, на рівні спілкування, замовника (проектувальника) із представниками виробництва недостатньо, щоб запевнити обидві сторони у відсутності або присутності хибних даних у програмному забезпеченні, апаратних чи конструктивних проблем.

Стандартна система мікроконтролерного управління складається з самого мікроконтролера (МК) та периферії, яка включає датчики, сенсори та виконавчі механізми. Перевірка периферії здійснюється шляхом повірки та калібровки відповідних периферійних пристроїв з застосуванням спеціальної апаратури і це гарантує виробник зазначених пристроїв. В мікроконтролері в свою чергу можна виділити апаратну та програмну частини. Апаратна частина перевіряється спеціальними тестами виробниками відповідних мікроконтролерів і вони гарантують працездатність своїх виробів. Програмна частина мікроконтролерного пристрою керування (прошивка МК) реалізує алгоритм управління, і саме вона забезпечує відповідність специфікації всієї системи мікроконтролерного управління. Таким чином, проектувальник мікроконтролерної системи управління розробляє саме програмну частину проекту та тести для неї (сценарії тестування програмної частини) і передає це виробнику. Виробник перевіряє виготовлений пристрій на тестах проектувальника і повертає йому результат тестування, що підтверджує працездатність виготовленого пристрою.

Метою проєкту є імплементація методу тестування цифрових пристроїв потужностями виробництва за сценаріями замовника,

узгодженими з обох сторін. Для імплементації методу тестування використовується будь-який із чотирьох сценаріїв, опис яких наведений у підрозділі 1.2. даної роботи, інформаційної та комунікативної взаємодії замовника та виробника вбудованих мікроконтролерних систем. Основою є використання тестової специфікації, яка дещо відрізняється від стандартної, із необхідним програмним забезпеченням для тестування всіх важливих апаратних компонентів та алгоритмів роботи із подальшим збереженням результатів тестування та їх відображенням у вигляді конструктивних особливостей пристрою на стороні виробника (світло, звук, текст).

Тестова специфікація (ТС) – це документ, який має в собі всю необхідну інформацію, а саме: таблицю з інформацією, коли, ким та з якої причини був змінений даний документ (release note), перелік необхідних апаратних компонентів, які мають бути використані у тестуванні (prerequisites), опис програмного забезпечення, яке має бути завантажене на апаратне забезпечення та як його завантажувати (software guide), покроковий опис виконання «позитивного» тестування (test execution) і опис помилок, які можуть виникнути при тестуванні (error codes).

Програмне забезпечення (ПЗ) – це мінімальний, критично-важливий функціонал для виконання тестування за специфікацією, а також збереження інформації будь-якого результату тесту. На стороні виробника кількість збереженої інформації обумовлена тільки конструктивними особливостями пристрою, оскільки дані, які мають бути збережені для замовника можуть бути конфіденційними та порушувати узгоджений сценарій взаємодії (рис 1.6).

Запропонований сценарій взаємодії замовника (розробника) з потужностями виробництва має бути реалізований на прикладі проектування та тестування мікроконтролерної системи логічного управління з передачею відповідної діагностичної інформації замовнику. Результат може бути наданий у формі відео тестування або ж надання тест-ріпорту.

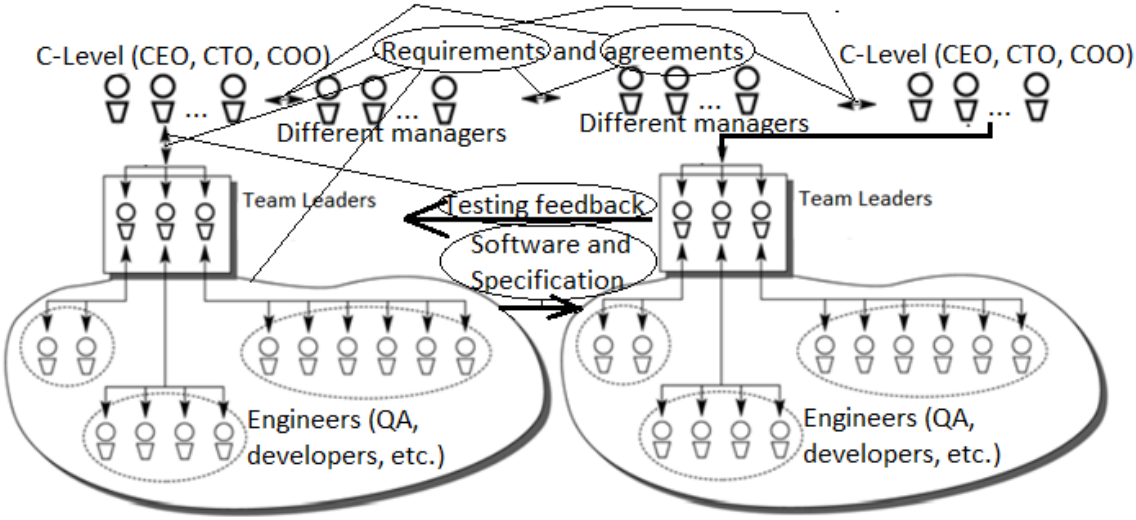


Рисунок 1.6 – Метод інформативної та комунікативної взаємодії замовника та виробника за сценаріями п.1.3

2 ТЕХНОЛОГІЇ ДІАГНОСТУВАННЯ МІКРОКОНТРОЛЕРНИХ ПРИСТРОЇВ

2.1 Методи діагностування мікроконтролерних систем

Під діагностуванням розуміється процес пошуку та виявлення в розробленій та налагодженій системі помилок, викликаних несправностями в апаратному забезпеченні.

Як уже зазначалося раніше, ненадійність програмного забезпечення мікроконтролерів пояснюється наявністю в ньому помилок, внесених на стадії проектування. Під час налагодження надійність програмного забезпечення підвищується.

На етапі експлуатації мікроконтролерних систем для забезпечення можливості реалізації ними цільової функції необхідно виявляти факти помилок у їх роботі, визначати місце виникнення помилки та її причину і робити ліквідацію помилки та, можливо, її наслідків. Часто повторювані помилки, визвані недостатньо якісним проектуванням системи доцільно усувати шляхом перепроектування системи чи її окремих вузлів.

Розрізняють функціональне та тестове діагностування систем. Функціональне діагностування здійснюється в процесі використання системи за її цільовим призначенням, при цьому на входи системи подаються лише робочі сигнали. Тестове діагностування проводиться шляхом переведення системи в особливий режим, при цьому на входи системи подаються спеціальні тестові сигнали.

Для проведення діагностування застосовуються програмні, апаратні та програмно-апаратні засоби.

Програмні засоби діагностування, що є частиною системного ПЗ, є резидентними програмами. Під час проведення діагностування програмними засобами слід пам'ятати, що достовірність діагностування залежить від

справності певної частини мікроконтролерної системи, достатньої щодо діагностування.

Тест виконавчого блоку мікроконтролера призначений виявлення несправностей механізмів управління та обробки даних блоку виконання команд. Тест реалізується шляхом виконання команд із заздалегідь визначеним результатом їх виконання та подальшого порівняння обчислених результатів з еталонними. В якості результатів в залежності від типу команди, що перевіряється можуть виступати вміст комірки пам'яті, прапори слова стану програми, адреса переходу тощо. Такі тести можуть складатися з детермінованої послідовності команд або являти собою псевдовипадкові генеровані послідовності. Однією з проблем при організації тестування блоку є проблема виявлення чутливості до певних послідовностей команд.

Тести пам'яті (ПЗП і ОЗП) призначені виявлення факту спотворення інформації, що зберігається в пам'яті, через несправності, пов'язані з дефектами фізичної структури елементів пам'яті (пробої на шини живлення, взаємний вплив осередків та ін.).

Тест ПЗП реалізується за допомогою визначення контрольної суми його вмісту та її подальшого порівняння з еталонною контрольною сумою, що також зберігається в ПЗП, отриманої перед занесенням фінальної версії програмного забезпечення в пам'ять програм. У найпростішому випадку контрольна сума може бути сумою по модулю 256 всіх байтів пам'яті програм, розташованих в області пам'яті, що підлягає контролю.

Тест оперативної пам'яті (ОП) реалізується за допомогою запису в кожну з осередків тестового слова, що підлягають контролю, наступного читання вмісту кожної комірки та порівняння її фактичного вмісту та значення занесеного до неї тестового слова. В якості тестових комбінацій можуть виступати: довільні дані, шаховий код (чергування в слові, що записується, нулів і одиниць, наприклад, для осередків ємністю один байт це коди 5516 і AA16), коди "нуль, що біжить" і "одиниця що біжить", при проведенні яких для комірки шириною m біт виробляється m циклів "запис-

читання" (наприклад, для комірки ємністю один байт тест "біжить нуль" являє собою вісім кодів: 11111110, 11111101, 11111011, …, 1011111). Для тестування ОЗП застосовуються інші, складніше сформовані послідовності.

Тести периферійних вузлів призначені для виявлення несправностей у спеціалізованих модулях мікроконтролера: портах, аналого-цифрових перетворювачах та ін. Такі тести, як правило, вимагають використання додаткових апаратних засобів для фіксації вихідних сигналів мікроконтролера, що генеруються тестом, або для подачі на нього еталон. При тестуванні портів введення-виведення використовуються комутатори, що дозволяють подати на розряди портів, що використовуються для введення інформації, сигнали з розрядів портів, що використовуються для виведення інформації. При тестуванні аналого-цифрових перетворювачів використовуються джерела опорних напруг.

Коди з виявленням помилок призначені підвищення надійності функціонування як мікроконтролерної системи загалом, і її окремих частин, наприклад, послідовних каналів зв'язку й оперативної пам'яті. Основна ідея використання таких кодів полягає у запровадженні інформаційної надмірності, що полягає у застосуванні додаткових службових розрядів, зміст яких залежить від інформаційних. При записі коду з виявленням помилок службові розряди формуються за певними правилами як від інформаційних розрядів. Під час читання коду відбувається повторне формування службових розрядів та його порівняння зі сформованими раніше. При використанні достатньої інформаційної надмірності можливе не тільки виявлення факту помилки, а й визначення її розташування та виправлення.

Як приклад застосування коду з виявленням помилок у мікроконтролерних системах наведемо другий і третій режими роботи послідовного порту мікроконтролера МК-51, при використанні яких на додаток до восьми інформаційних розрядів передається дев'ятий біт, якому може бути програмно поставлено у відповідність значення біта парності кількості одиниць у переданій послідовності.

Апаратні засоби діагностування є технічні пристрої, що дозволяють проводити пошук відмов у обчислювальній системі. На додаток необхідно відзначити їх важливе значення при неможливості застосування більш зручних у застосуванні програмних засобів діагностування (наприклад, через несправності ядра мікроконтролерної системи). Зазначені апаратні засоби діагностування дозволяють визначити параметри напруги живлення, наявність їм

Засоби тестування навантаженнями призначені для навмисного зміни параметрів довкілля з метою аналізу впливу цієї зміни на роботу мікроконтролерної системи. Застосовуються три різновиди такого тестування – механічні, температурні та електричні навантаження.

До програмно-апаратних засобів діагностування відносяться дублюючі системи, пристрої зберігання тестів-еталонів, контролюючі автомати.

У зв'язку з тим, що експлуатація резервованих мікроконтролерних систем здебільшого невиправдана (у зв'язку з погіршенням масогабаритних, вартісних та інших характеристик) системи дублювання застосовуються лише при проведенні тестового діагностування.

Одним з варіантів побудови системи дублювання є застосування спеціального пристрою для зберігання ненадлишкових тестових послідовностей. У даному варіанті порядок слідування тестових сигналів визначається пристроєм управління виходячи з обраного користувачем режиму тестування та результатів порівняння реакцій еталонної та діагностованої систем на попередніх кроках тесту (наприклад, проведення розширеного тестування для точного визначення характеру несправності при розбіжності результатів).

Другий клас програмно-апаратних засобів діагностування – системи зберігання тестів-еталонів, що є логічним розвитком описаної вище структури. У разі еталонні вихідні сигнали генеруються не еталонної системою, а зберігаються у пам'яті поруч із вхідними тестовими сигналами. Такі системи в порівнянні з системами дублювання мають кращі вартісні та

масогабаритні характеристики, а також знімають проблему можливих появ несправностей в еталонній системі.

До третього класу програмно-апаратних засобів діагностування відносяться контрольні автомати. Контролюючі автомати будуються для контрольованого автомата А та відомого класу помилок у ньому, що підлягають виявленню у вигляді автомата A_E. Помилкою в автоматі А є спотворення його вихідних сигналів, а також його внутрішнього стану Q (слід зазначити, що спотворення Q може не впливати на протягом деякого часу, тобто для зовнішнього спостерігача діагностована система може продовжувати функціонувати достовірно). Завданням контролюючого автомата є виявлення будь-якої помилки в контрольованому автоматі в момент її виникнення, у зв'язку з чим на вхід автомата АЕ подаються не лише входи Х та виходи У

2.2 Особливості тестування мікроконтролерних програм

Тестування програмного забезпечення область дуже складна та велика. З цієї теми написано багато робіт стосовно різних методів розробки ПЗ та стеків технологій. Однак, більшість з них присвячена тестуванню серверного та прикладного ПЗ для ПК. Тема тестування мікроконтролерів освітлена вкрай мало.

Насамперед тестування потрібно для того, щоб упевнитися у працездатності та відповідності певним вимогам програми в цілому та/або її окремих частин. Звичайно, наявність тестів, сама по собі не може гарантувати повну відсутність помилок у системі, що розробляється, проте тести істотно підвищують ймовірність раннього виявлення помилок, скорочують час їх пошуку. Як відомо, помилка під час компіляції не коштує нічого, виявлена під час первинного тестування – майже нічого (час розробника/тестувальника), виявлена під час альфа/бета тестування – коштує

мало, а ось критична помилка знайдена після серійного випуску продукту може бути дуже дорогою.

Таким чином, основне завдання тестування (не тільки ПЗ) можна сформулювати так: якомога раніше виявлення помилок, щоб заощадити гроші і час.

Чим відрізняється тестування програм МК від тестування прикладного чи серверного ПЗ для ПК?

На ПК усі доріжки вже давно проторені (ну основні точно): операційні системи (ОС) відносно стандартні – Windows/Unix, у кожному стеку технологій розробки є свої бібліотеки для підтримки тестування, свої методи роботи з ними тощо.

Для МК все простіше та складніше одночасно. Простіше тому, що складність програм для МК, навіть для товстих ARM-ів, набагато менша, ніж для ПК. Складніше тому, що оточення, в якому працює програма, може бути дуже різноманітним. Програма має працювати з вбудованою периферією МК та з різними зовнішніми пристроями. Робота з великою різноманітністю заліза на низькому рівні в поєднанні з жорсткими вимогами швидкодії робить тестування непростим завданням.

Часто ПЗ МК тестують вручну методом «тестового прогону». Скомпільовану прошивку завантажують у цільову систему, або її макет, запускають і за допомогою якогось налагоджувального інтерфейсу/виводу (USART, світлодіоди, осцилограф, JTAG і т.д.) спостерігають за її роботою. Також вручну можна імітувати зовнішні впливи та змінювати по ходу виконання різні параметри. Це виходить ручне інтеграційне/системне тестування. Однак, якщо програма не працює або працює не правильно, зрозуміти в чому причина буває дуже складно. Налагоджувати програму повністю за кроками дуже довго і не продуктивно. Сюди ж можна віднести випробування на універсальних симуляторах типу Proteus і подібних до нього, де симулюється не тільки МК з внутрішньою периферією, але і зовнішня обв'язка в цілому.

У більшості випадків програма МК виконує багато різних функцій, які можна виділити більш-менш незалежно від інших, такі як виведення на дисплей, опитування клавіатури, управління двигуном тощо. Багато розробників проектують такі функції окремо у спеціальних тестових проектах, а потім копіюють готовий налагоджений код у цільовий проект. Це свого роду "ручне модульне тестування".

Потім цей «налагоджений» код переноситься в наступний проект, але оскільки він, швидше за все, не зовсім відповідає новому завданню, то він модифікується на місці під нові вимоги. Адже будь-яка навіть найакуратніша зміна може загрожувати помилками. В ідеалі після кожної істотної зміни код має бути заново протестований. Вручну це робити вкрай трудомістко і багато хто невиправдано відмовляється від частого модульного тестування на користь системного знову-таки ручного тестування.

На допомогу приходить автоматизація. Більшість тестів як модульних, і системних можна автоматизувати тими чи іншими способами. Для платформо-незалежного коду, різних контейнерів, типу списків, черг, дерев і тощо, високорівневих протоколів обміну, кінцевих автоматів та інші. Цей процес легко автоматизується, запуск одного скрипта або програми дозволяє швидко перевірити такі модулі на наявність помилки.

З платформо-залежним кодом дещо складніше. Його звичайно потрібно тестувати в цільовому оточенні. Можна, звичайно, проводити тестування безпосередньо на цільовій платформі з необхідним зовнішнім обважуванням та налагоджувальним обладнанням, але автоматизувати цей процес набагато важче, ніж для платформо-незалежних модульних тестів. Цей підхід дуже важко і краще підходить для комплексного системного тестування, ніж для модульних тестів. Як варіант, можна спробувати використовувати універсальні симулятори (той самий Proteus), але вони дуже ресурсозалежні і автоматизуються дуже погано.

Поки що все виглядає так, ніби всеосяжне автоматизоване тестування – це альтернатива, що дозволяє відловити, якщо не всі, то майже всі дефекти в

програмі, що розробляється. У реальному житті це не так. По-перше, якщо просто наклепати будь-яких тестів для вже написаної програми, то це не дасть зовсім нічого. Такі тести просто показуватимуть, що програма працює саме так, як вона працює і будуть придатні лише для перевірки, що нові зміни не зламали стару функціональність (вже корисно, в принципі). По-друге, далеко не всякий код можна і потрібно тестувати незалежно від решти системи. Наприклад, драйвера різних специфічних зовнішніх пристроїв, які не можна симулювати, мабуть, можна тестувати тільки на реальному устаткуванні. Простіше кажучи, щоб протестувати фрагмент коду (модуль) потрібно у тестовому оточенні відтворити всі його вихідні залежності. Кожна зовнішня (по відношенню до тестованого коду) функція, кожна глобальна змінна є вихідною залежністю. Тут можна запровадити поняття «шва». Шов – це код (вихідна залежність), поведінка якого можна змінити, не змінюючи сам код.

```
volatile int value;
...
void Foo()
{
        // зробити якісь нетривіальні обчислення
        int bar = (value * 10) << 1024;
        // Перетворити результат у рядок
char buffer [9];
itoa(bar, buffer, 10);
// вивести його на LCD
lcd_home();
lcd_puts(buffer);
}
…
// десь використовуємо цю функцію
Foo();
```

Рисунок 2.1 – Приклад опису «шва»

Тут функція Foo має чотири залежності і всі вони не є швами: глобальна змінна value і функції itoa, lcd_home, lcd_puts. У такому вигляді ця функція практично нетестована, оскільки складно гарантувати певне значення глобальної змінної – воно може змінюватись у перериванні чи іншому потоці. Тобто потрібно тестувати і весь код, що використовує value.

Також важко проконтролювати результат, оскільки функція виведення lcd_puts жорстко задана. Спробуємо зробити деякі залежності у цьому прикладі «швами»:

```
int value;
void Foo (int arg, void (*print_func) (char *))
{
// зробити якісь нетривіальні обчислення
        int bar = (arg * 10) << 10;
// Перетворити результат у рядок
char buffer [9];
itoa(bar, buffer, 10);
// вивести результат кудись
print_func(buffer);
}
…
// Виведення рядка на LCD
void lcd_print(char *str)
{
lcd_home();
lcd_puts(str);
}
…
// десь використовуємо цю функцію
Foo(value, lcd_print);
```

Рисунок 2.2 – Використання залежностей як «швів»

Тепер у нас лише одна залежність – стандартна функція itoa, вона нам не заважає. Щоб протестувати нашу функцію, тепер достатньо написати заміну для lcd_print, яка перевірятиме переданий аргумент:

```
        void test_print(char * str)
        {
                const char *expected = "1234";
                if(strcmp(expected, str) != 0)
                {
                    fprintf(stderr, "Foo test failed. Expected %s, got %s", expected,
str);
                     exit(1);
                }
        }
        void testFoo()
        {
                Foo(126362);
                fprintf(stderr, "Foo test is OK");
        }
```

Рисунок 2.3 – Заміна функції

Цей тест не залежить від платформи і його можна успішно запускати на ПК. З цього прикладу видно, що чим менше у фрагмента коду вихідних залежностей, які не є швами, тим простіше та ефективніше його тестувати. Звідси висновок – програми спочатку потрібно розробляти з урахуванням потреби у тестуванні.

Проте, як видно з цього прикладу, тестування не завжди дається безкоштовно – у другому фрагменті на один виклик функції більше, причому це виклик за вказівником.

3 АПАРАТНА РЕАЛІЗАЦІЯ ТЕХНОЛОГІЇ ДІАГНОСТУВАННЯ МІКРОКОНТРОЛЕРНИХ ПРИСТРОЇВ

3.1 Сімейство плат розробки Arduino

Arduino — це платформа прототипування з відкритим вихідним кодом, заснована на простому у використанні апаратному та програмному забезпеченні. Плати Arduino здатні зчитувати вхідні дані з датчиків або кнопок, обробляти їх і перетворювати на вихід.

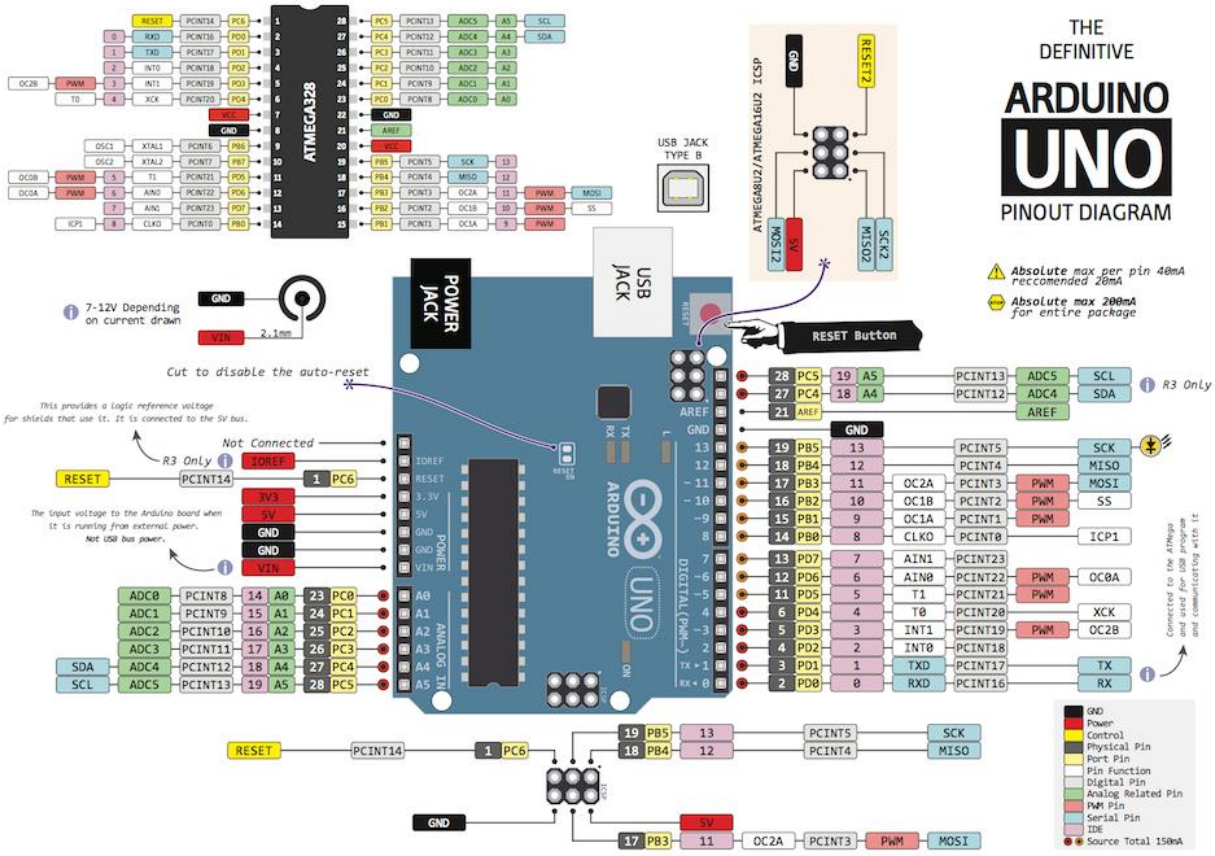


Рисунок 3.1 – Плата Arduino UNO

Arduino – це плата розробки мікроконтролерних рішень із самим, власне, мікроконтролером ATmega328p, що працює на частоті 16 МГц. Складається з друкованої плати, яку можна програмувати (її називають мікроконтролером), і готового програмного забезпечення під назвою Arduino

IDE (Integrated Development Environment – інтегроване середовище розробки), яке використовується для написання та завантаження комп'ютерного коду на фізичну плату.

Існує багато типів Arduino: Arduino Uno (рис.3.1), Arduino Mega, Arduino Nano, Arduino Pico, Arduino Due (84MHz), Arduino Leonardo тощо.

На прикладі Arduino UNO та Arduino Nano (найпоширеніших та найпопулярніших плат розробки даного сімейства), розглянемо основні функції портів введення-виведення та роботи плати загалом. Номінальна напруга для роботи плати складає 5В (рис.3.2). Має 14 цифрових контактів GPIO (General Purpose Inputs Outputs – порти введення-виведення) і 6 аналогових. Роз'єми живлення 5В і 3,3 В з 3(трьома) контактами заземлення на борту. Ще один мікроконтролер SMD, вбудований для програмування основного чіпа Arduino. Є також два регулятори напруги: один на 5 В, а інший на 3,3 В. для протоколу SPI (Serial Peripheral Interface – Послідовний Інтерфейс Периферії) є піни MISO (Main Input Secondary Output), MOSI (Main Output Secondary Input), SCK (Synchro Clock) і CS(Chip Select). Для протоколу I2C є контакти SCL (Synchro Clock) і SDA (Serial Data). Ви також можете підключити пристрій до послідовного зв'язку (UART – Universal Asynchronous Receive Transmit) за допомогою контактів Rx і Tx. На платі також є кілька контактів ШІМ (Широтно-імпульсної модуляції).

Arduino — це апаратна обчислювальна платформа, основними компонентами якої є порти введення/виведення та середовище розробки на мові Processing/Wiring. Arduino можна використовувати як для створення автономних інтерактивних об'єктів, так і для підключення до програмного забезпечення, що працює на настільному комп'ютері (наприклад: Adobe Flash, Processing, Max/MSP, Pure Data, SuperCollider).

На базі Arduino навчальні приклади, роботи, системи спостереження та безпеки, аналоги систем «розумний дім», гірлянда з «вогниками, що біжать», цифровий кодовий замок, пульт керування побутовою технікою, система автоматичного поливу. квітів тощо.

Усі плати можна програмувати через USB (Universal Serial Bus – універсальна послідовна шина), що стало можливим завдяки мікросхемі-перетворювача USB-to-Serial. У версії платформи Arduino UNO, в якості перетворювача використовується контролер Atmega16U2 в корпусі SMD. Це рішення дозволяє запрограмувати конвертер таким чином, щоб платформа відразу розпізнавалася як мишка, клавіатура та інший пристрій на вибір розробника з усіма необхідними додатковими сигналами управління. Деякі варіанти, такі як Arduino Mini, потребують підключення окремого перетворювача USB-to-Serial до контролера для програмування.

Pin Category	Pin Name	Details
Power	Vin, 3.3V, 5V, GND	Vin: Input voltage to Arduino when using an external power source. 5V: Regulated power supply used to power microcontroller and other components on the board. 3.3V: 3.3V supply generated by on-board voltage regulator. Maximum current draw is 50mA. GND: ground pins.
Reset	Reset	Resets the microcontroller.
Analog Pins	A0 – A5	Used to provide analog input in the range of 0-5V
Input/Output Pins	Digital Pins 0 - 13	Can be used as input or output pins.
Serial	0(Rx), 1(Tx)	Used to receive and transmit TTL serial data.
External Interrupts	2, 3	To trigger an interrupt.
PWM	3, 5, 6, 9, 11	Provides 8-bit PWM output.
SPI	10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)	Used for SPI communication.
Inbuilt LED	13	To turn on the inbuilt LED.
TWI	A4 (SDA), A5 (SCA)	Used for TWI communication.
AREF	AREF	To provide reference voltage for input voltage.

Рисунок 3.2 – Опис портів введення-виведення плати розробки UNO, Nano

Microcontroller	ATmega328P – 8 bit AVR family microcontroller
Operating Voltage	5V
Recommended Input Voltage	7-12V
Input Voltage Limits	6-20V
Analog Input Pins	6 (A0 – A5)
Digital I/O Pins	14 (Out of which 6 provide PWM output)
DC Current on I/O Pins	40 mA
DC Current on 3.3V Pin	50 mA
Flash Memory	32 KB (0.5 KB is used for Bootloader)
SRAM	2 KB
EEPROM	1 KB
Frequency (Clock Speed)	16 MHz

Рисунок 3.3 – Характеристики плат розробки UNO, Nano

Arduino складається з мікроконтролера Atmel, а також елементів для програмування та інтеграції з іншими пристроями. Багато плат мають лінійний стабілізатор напруги +5 В або +3.3В. Тактування здійснюється на частоті 84 МГц, 16 МГц або 8 МГц кварцовим резонатором.

3.2 Сімейство мікроконтролерів STM

Широкий асортимент мікроконтролерів ST охоплює від надійних недорогих 8-розрядних мікроконтролерів до 32-розрядних мікроконтролерів Cortex-M на базі Arm із широким вибором периферійних пристроїв. Наявність широкого асортименту гарантує, що інженери-конструктори знайдуть поєднання продуктивності, енергоефективності та безпеки, яке вимагається для їх застосування.

Платформа 8-розрядного мікроконтролера ST реалізована навколо високопродуктивного 8-розрядного ядра та найсучаснішого набору

периферійних пристроїв. Ця платформа виготовлена за власною технологією вбудованої енергонезалежної пам'яті ST розміром кристалу у 130 нм.

STM8 забезпечує швидку та безпечну розробку за допомогою розширених операцій вказівника стека, розширених режимів адресації та нових інструкцій.

Платформа 8-бітного мікроконтролера підтримує чотири серії продуктів: STM8S – «мейнстрімна» або «базова» серія, STM8L – енергоефективні та STM8AF з STM8AL – автомобільні мікроконтролери серій «мейнстрім» та «енергоефективність» (рис.3.4).

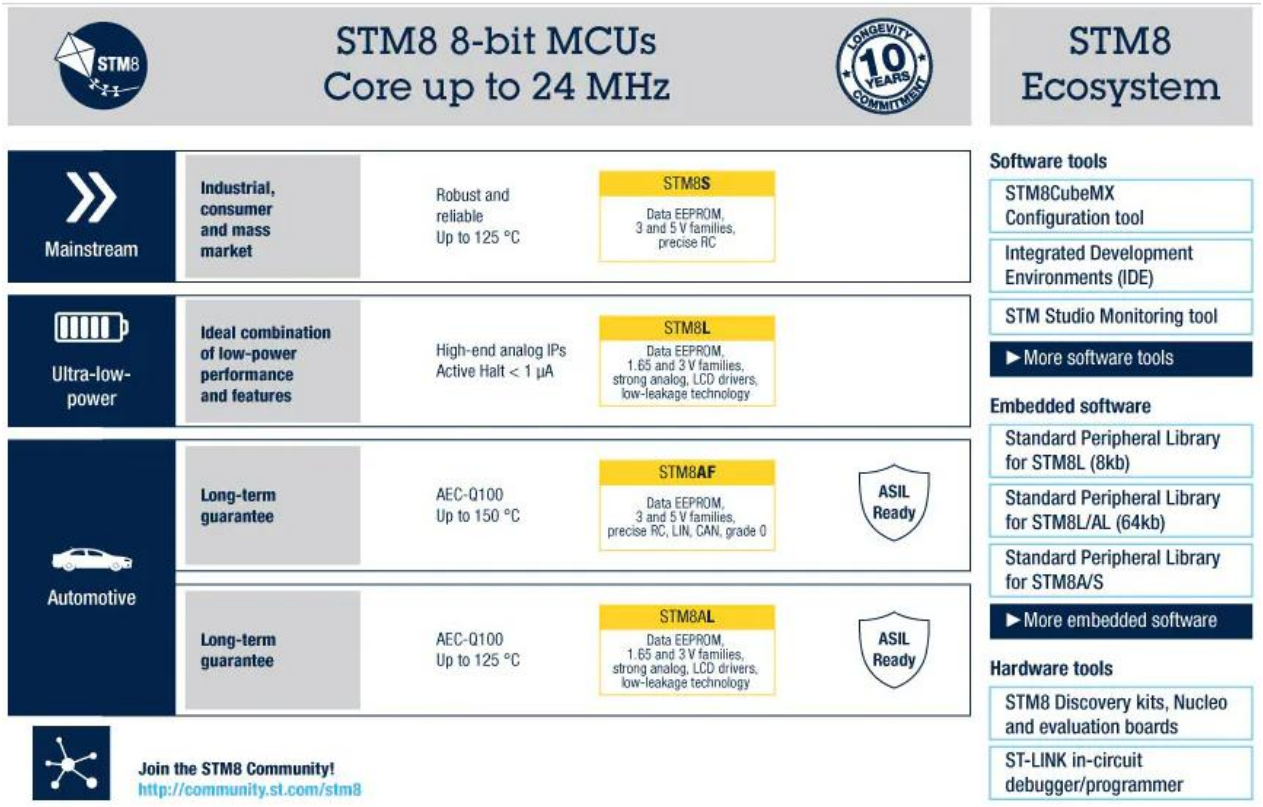


Рисунок 3.4 – Презентація застосування 8-бітного сімейства STM

Масштабні рішення, маленькі та великі проєкти з різноманітністю функціоналу та робота з операційними системами реального часу (ОСРВ), потребують не тільки обробки більших даних, але й за швидший час. У світ мікроконтролерів виходить сімейство 32-розрядних STM32 на основі процесора ARM (Advanced RISC Machine) Cortex-M розроблено, щоб надати користувачам MCU (microcontroller unit) нові ступені свободи. Він пропонує

продукти, що поєднують дуже високу продуктивність, можливості в режимі реального часу, цифрову обробку сигналів, роботу з низьким енергоспоживанням/низькою напругою та підключення, зберігаючи повну інтеграцію та легкість розробки. Лінійка мікроконтролерів STM32, заснована на ядрі промислового стандарту, поставляється з широким вибором інструментів і програмного забезпечення для підтримки розробки рішень.

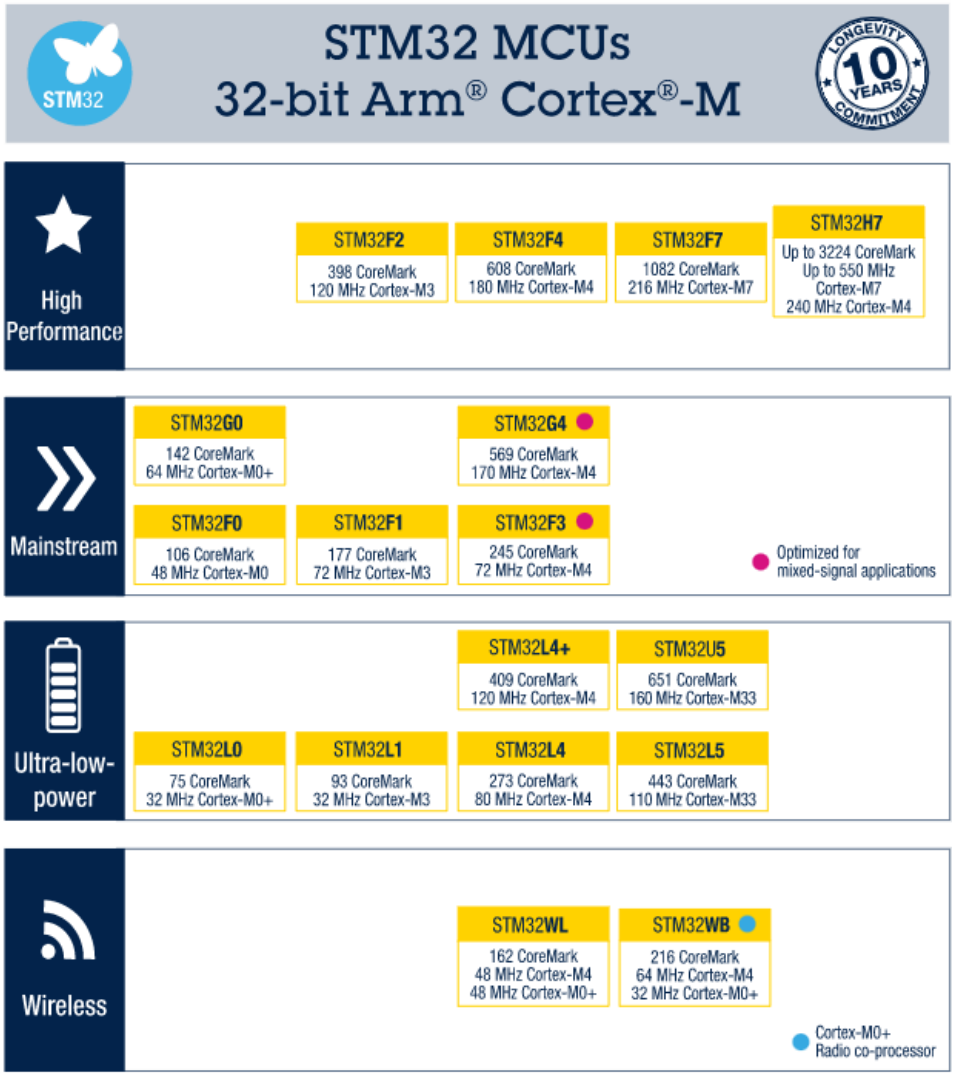


Рисунок 3.5 – Презентація застосування 32-бітного сімейства STM

На прикладі найпопулярнішого підсімейства (серії) мікроконтролерів STM32F1xx, розглянемо характеристики та периферію, яку підтримують виключно усі сімейства.

Серія стандартних мікроконтролерів STM32F1 від ST задовольняє потреби широкого спектру рішень на промислових, медичних і споживчих

ринках. З цією серією ST стала піонером у світі мікроконтролерів Arm Cortex-M і встановила віху в історії вбудованих програм. Висока продуктивність із першокласними периферійними пристроями, енергоефективність з низькою напругою працює в поєднанні з високим рівнем інтеграції за доступними цінами з простою архітектурою та простими у використанні інструментами.

Серія складається з п'яти ліній продуктів, сумісних за технологією «pin-to-pin», периферійними пристроями та програмним забезпеченням (рис.3.6).

1. STM32F100 Value line – процесор з 24 МГц з керуванням двигуном і функціями CEC
2. STM32F101 – процесор з 36 МГц, флеш-пам’ять до 1 Мбайт
3. STM32F102 – процесор з 48 МГц з USB FS
4. STM32F103 – процесор з 72 МГц, до 1 Мбайт флеш-пам’яті з керуванням двигуном, USB та CAN. Являється найпопулярнішим мікроконтролером.
5. STM32F105/107 – процесор з 72 МГц та Ethernet MAC, CAN і USB 2.0 OTG

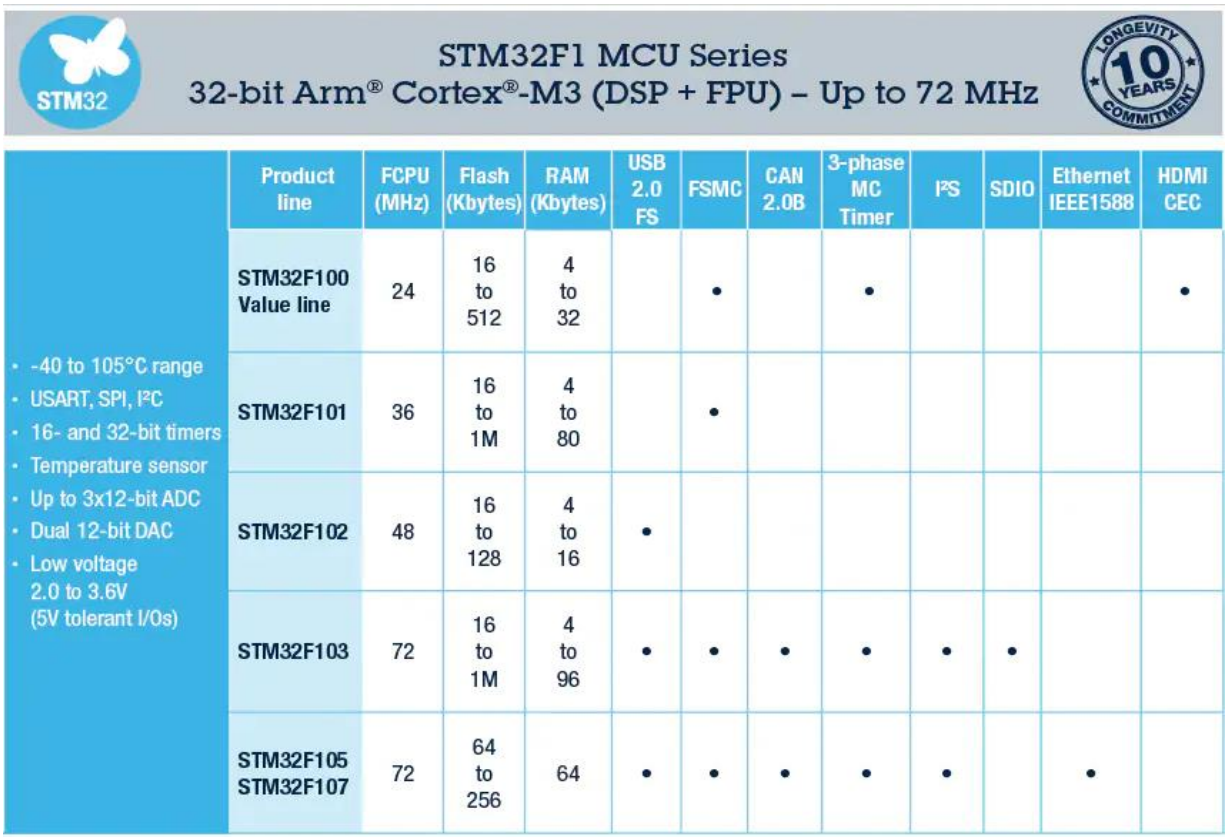
Сімейство високопродуктивних лінійок STM32F103xx із середнім набором функціоналу та характеристик включає високопродуктивне 32-розрядне ядро RISC ARM Cortex-M3, що працює на частоті 72 МГц, високошвидкісну вбудовану пам’ять (флеш-пам’ять до 128 Кбайт і SRAM до 20 Кбайт), а також широкий спектр розширених пристроїв введення/виведення та периферії, підключених до двох паралельних шин APB (Advanced Peripheral Bus). Усі пристрої пропонують два 12-бітних АЦП(Аналого-цифрових перетворювача – ADC), три 16-бітних таймера загального призначення плюс один ШІМ-таймер, а також стандартні та розширені інтерфейси зв’язку: до двох I2C та SPI, три USART, USB та CAN (рис.3.7).

Пристрої працюють від джерела живлення в діапазоні від 2,0В до 3,6 В.

Вони доступні як для температурного діапазону від –40°C до +85°C, так і для розширеного температурного діапазону від –40°C до +105°C. Комплексний набір режимів енергозбереження дозволяє розробляти програми з низьким енергоспоживанням.

Лінійка STM32F103xx включає пристрої в шести різних типах корпусу: від 36 до 100 контактів.

Ці особливості роблять лінійку мікроконтролерів STM32F103xx придатною для широкого спектру рішень, таких як електроприводи, керування, медичне та портативне обладнання, периферійні пристрої для ПК та ігор, платформи GPS, промислові програми, ПЛК (програмовані логічні контролери – PLC), інвертори, принтери, сканери, сигналізації, відеодомофони та ОВК (опалення, вентиляція та кондиціонування повітря – HVAC).



Product line	FCPU (MHz)	Flash (Kbytes)	RAM (Kbytes)	USB 2.0 FS	FSMC	CAN 2.0B	3-phase MC Timer	I²S	SDIO	Ethernet IEEE1588	HDMI CEC
STM32F100 Value line	24	16 to 512	4 to 32		•		•				•
STM32F101	36	16 to 1M	4 to 80		•						
STM32F102	48	16 to 128	4 to 16	•							
STM32F103	72	16 to 1M	4 to 96	•	•	•	•	•	•		
STM32F105 STM32F107	72	64 to 256	64	•	•	•	•	•		•	

Рисунок 3.6 – Опис периферії «мейнстрімного» підсімейства мікроконтролерів STM32F1xx

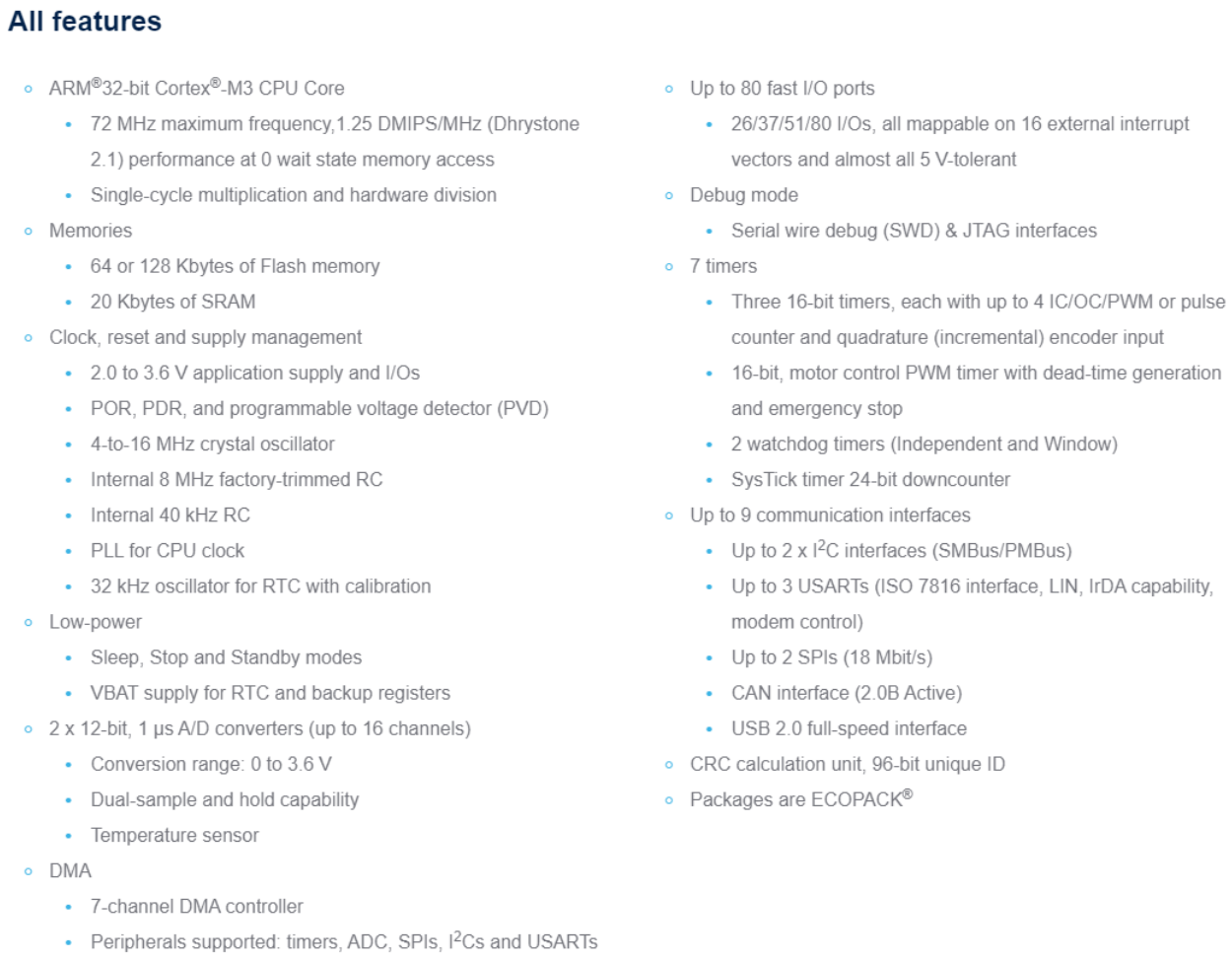


Рисунок 3.7 – Характеристики лінійки STM32F103

3.3 Апаратні особливості організації тестування мікроконтролерів

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

Відлагоджувальні 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 відлагоджувальної плати залежить від функціональних можливостей.

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

В спрощеному варіанті замість ПК може використовуватися стендова мікроконтролерна плата (плата для тестування, ПДТ), яка через відповідній інтерфейс зв’язку взаємодіє з мікроконтролерною платою, що тестується (ТП), що зображено на рис. 3.8.

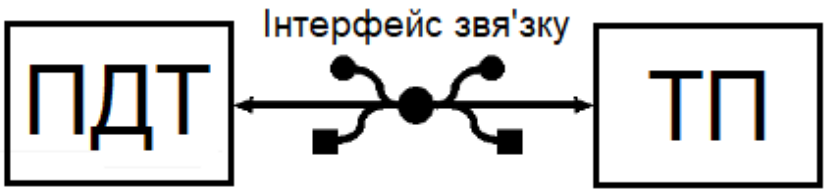


Рисунок 3.8 – Схема апаратного тестування МК

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 ПДТ і ТП 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 плати, що тестує до плати, що тестується і навпаки.

Якщо ж необхідно підключити різні мікроконтролери будь-якого спектру (різні сімейства, серії або ж лінійки), варто ще до розробки рішення виокремити характеристики обраних пристроїв управління (ПУ). Такими характеристиками являються:

1. Узгодження рівнів. ПДТ та ТП мають бути узгоджені за рівнем напруги. ПДТ може мати менший рівень напруги для роботи, проте входити в межі напруги роботи ТП. 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. Накопичення даних та їх відправка від ПДТ до ТП за один раз не вплине ні на швидкість виконання операцій, ні на кількість даних, що обробляються. Однак збільшується час на збір та завантаження необхідної кількості даних від ПДТ.

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 ТП також мала 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 як для ПДТ, так і для ТП.

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, I2C, USB тощо). Протокол передачі даних може бути синхронним або асинхронним, послідовним або паралельним. Як і з третім пунктом узгодження дана периферія має бути реалізована на ТП та мати той самий логічний рівень передачі даних, що і ПДТ (див. п.1 узгоджень). Ті ж самі вимоги стосуються і «прямого» підключення ПДТ до ТП. Прямим підключенням вважається спосіб «pin-to-pin», коли GPIO плати, що тестує напряму підключені до плати, що тестується. Узгодження рівня напруги має бути ще до реалізації рішення.

Всі пункти, опис яких знаходиться вище, є найважливішими та не можуть бути не враховані. Якщо узгодження правил тільки між ПДТ та ТП неможливе, то існує підхід у використанні 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. Неконтрольоване використання додаткових апаратних рішень між ПДТ та ТП може призвести до ускладнення роботи плат, некоректності їх роботи або повної відмови алгоритму роботи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕХНОЛОГІЇ ДІАГНОСТУВАННЯ МІКРОКОНТРОЛЕРНИХ ПРИСТРОЇВ

4.1 Програмна реалізація тестування мікроконтролерів

Для взаємодії відлагоджувальних плат чи власних («кастомних») апаратних рішень необхідно мати алгоритм роботи, за яким реалізація рішення на мікроконтролері буде виконувати всі закладені функції ще на етапі проєктування. Таким алгоритмом являється програмне забезпечення (ПЗ), завдання якого – описати та імплементувати функціонал пристрою, що розробляється.

Імплементація програмного забезпечення – це набір команд, операцій, функцій тощо, який створює розробник за допомогою мови програмування. Даний набір має бути скомпільований чи інтерпретований у спеціальний файл (*.hex, *.bin тощо) для обраного мікроконтролера. Такий файл називається «прошивкою» і містить у собі всі необхідні дані, щодо пам’яті мікроконтролера, що та як має бути записане у пам’ять, які адреси не мають бути використані тощо.

Найчастіше для створення програмного забезпечення використовують спеціальний функціонал – програму, яка містить в собі компілятор чи інтерпретатор, бібліотеки системного та стандартного функціоналу, необхідні для опису програмного забезпечення, менеджер плат, в якому треба обрати необхідну апаратну реалізацію, додатковий функціонал у вигляді відлагодження плати, що використовується, комунікації з іншими пристроями за допомогою протоколів передачі даних тощо. Дані програми називаються інтегрованим середовищем розробки (Integrated Development Environment – IDE) та мають великий попит на використання не тільки у напрямку розробки вбудованих рішень, а й у створенні сайтів, кросс-платформенихрішень, відеоіндустрії, створенні рисунків та схем, 2D-3D

моделей тощо.

Програмна реалізація для плат Arduino створюється за допомогою Arduino IDE (рис.4.1). Досить легкий та вичерпний функціонал, який покриває усі потреби для опису алгоритму роботи з можливістю обрати необхідне сімейство чи лінійку мікроконтролерів, виставити необхідні для відлагодження налаштування через COM-порт – драйвер для з'єднання плати розробки із ПК з подальшим виведенням інформації на консоль. Мова програмування плат Arduino – мікс мови С та С++.

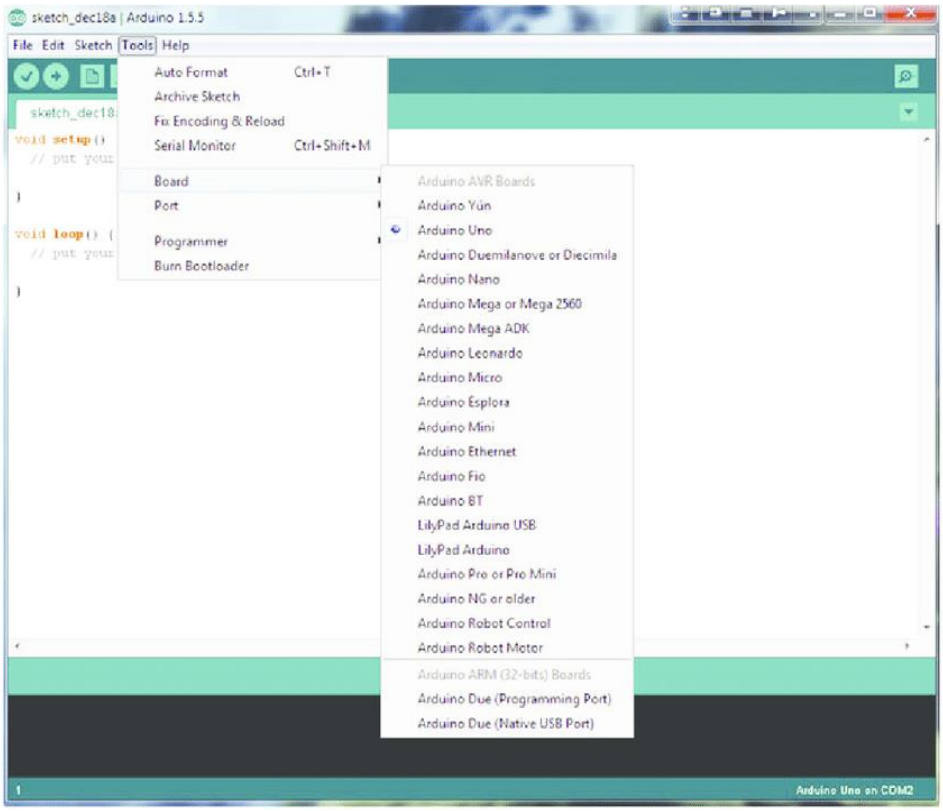


Рисунок 4.1 – Arduino IDE

Реалізація програмного рішення для плат сімейств чи лінійки STM32 виконується за допомогою двох середовищ. Перше середовище (рис.4.2) являє собою програму для конфігурації необхідної периферії (таймери, протоколи передачі даних, порти введення-виведення тощо), переривань, проєкту у цілому (який саме мікроконтролер використовується, ієрархія файлів у проєкті та як вони будуть створюватися). Друге середовище являє собою безпосередньо проєкт (рис.4.3), де підключені всі системні файли та

файли власної розробки. В ньому також існує великий функціонал з роботи як із файлами, так і з налаштуваннями проєкту для коректної та правильної прошивки мікроконтролера, з урахуванням рівня компіляції, кількістю необхідної пам’яті тощо. Перше середовище називається CubeMx, а друге – Keil uVision for ARM. Мовою програмування для мікроконтролерів STM32 є C.

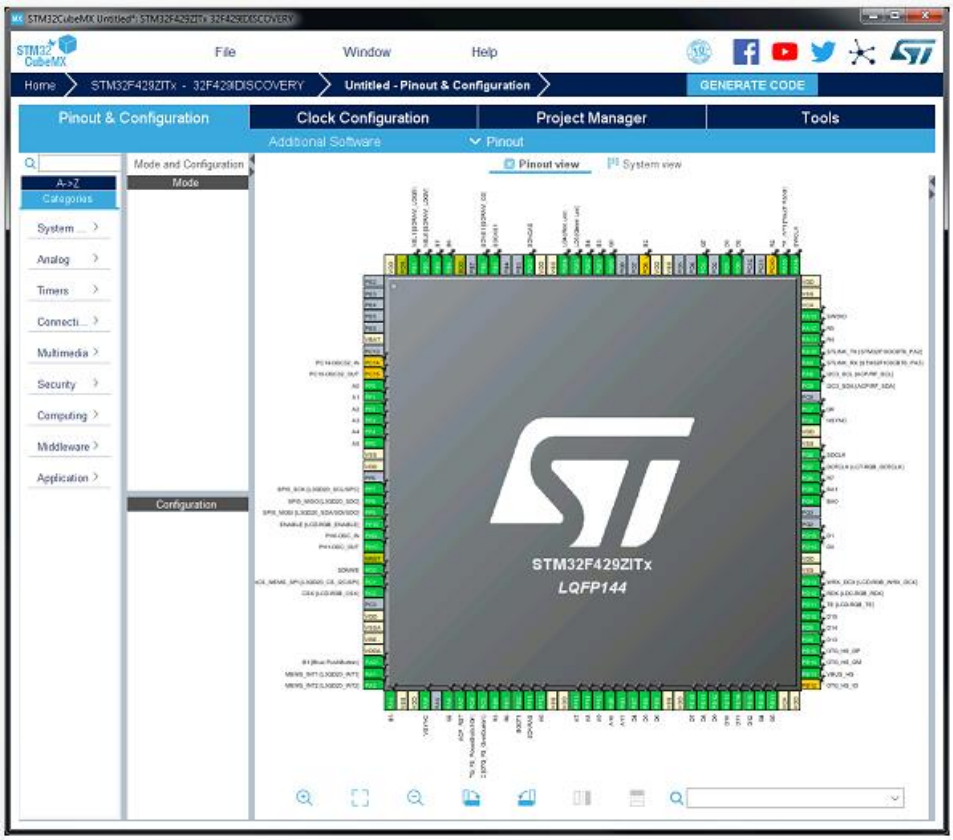


Рисунок 4.2 – CubeMX

Повна програмна реалізація тестування мікроконтролерів, опис якої наведено нижче складається з двох реалізацій. Перша є програмною реалізацією ПДТ та має керуючу властивість. Її опис зведено до видачі високого сигналу на конкретний пін введення-виведення. Видача високого сигналу відбувається після того, як ТП передасть значення готовності роботи. Це також відбувається видачею високого рівня сигналу на піні введення-виведення. Видача сигналу відбувається одразу як буде підключене живлення до ТП. Обмін даними між ПДТ та ТП буде відбуватися до тих пір,

допоки «позитивний» алгоритм ТП буде повністю відпрацьований, або ж відбудеться «негативний» результат сценарію (алгоритму) (рис. 4.5).

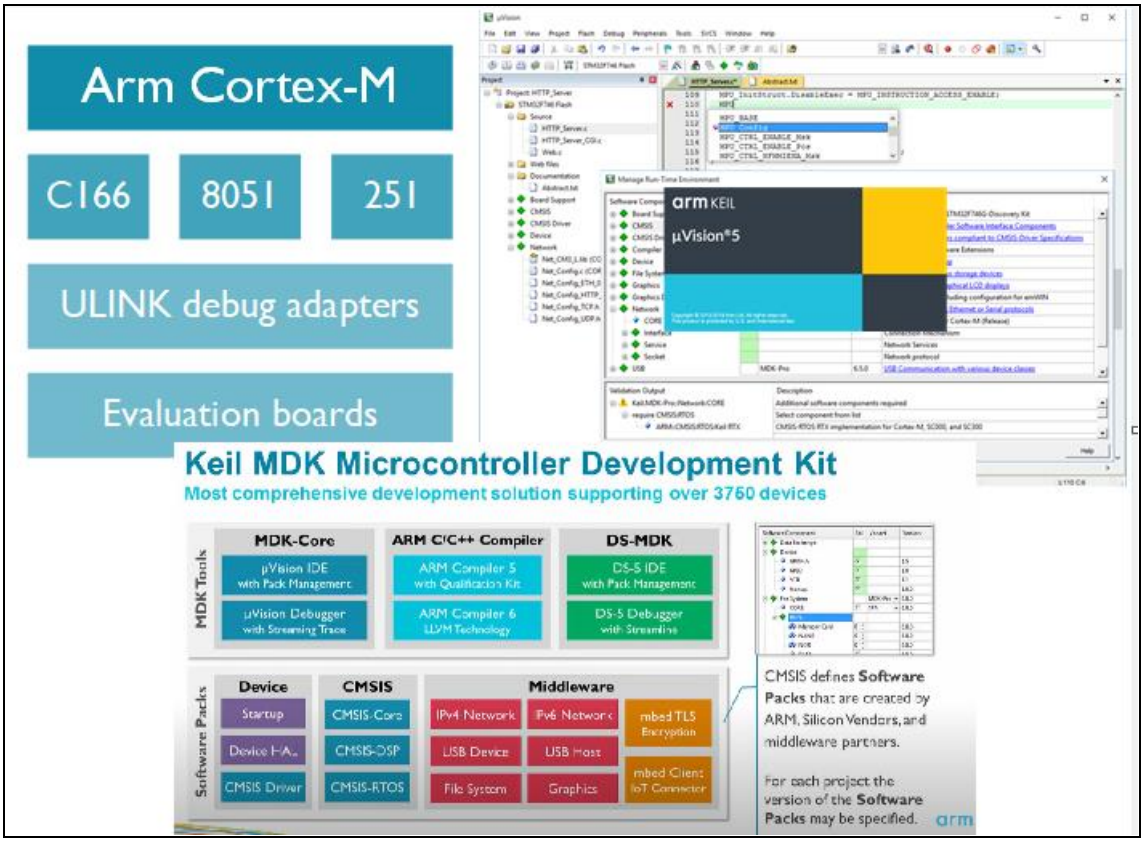


Рисунок 4.3 – Keil uVision

Друге програмне забезпечення являє собою алгоритм роботи ТП. Його опис полягає у передачі високого рівня сигналу на ПДТ в разі готовності отримати нову команду для реалізації тестового сценарію (рис.4.6). Коли ТП отримує високий рівень сигналу на власний пін введення-виведення від ПДТ, програмна реалізація крок за кроком виконує сценарій до повного його завершення, або ж до отримання «негативного» результату сценарію (рис.4.5).

```
if(HIGH==digitalRead(TEST_INPUT)){
  delay(10);
  if(HIGH==digitalRead(TEST_INPUT)){
    switch(testStep){
      case TEST_STEP_START:
        BoardON();
        testStep = TEST_STEP_PERIPH_CHECK;
      break;
      //Check the periph by software indication of hardware connecion
      //also check whether the memory able to write data
      case TEST_STEP_PERIPH_CHECK:
        #ifndef HWCONF_DHT11
          EEPROM.update(3,DHT11_ERR);
          ShowErrorEndTest(DHT11_ERR);
        #endif
        #ifndef HWCONF_EEPROM
          EEPROM.update(3,MEMORY_ERR);
          ShowErrorEndTest(MEMORY_ERR);
        #endif
        #ifndef HWCONF_RED_LED || HWCONF_GREEN_LED || HWCONF_BLUE_LED
          EEPROM.update(3,LED_ERR);
```

Рисунок 4.5 – Основний алгоритм роботи ТП

4.2 Інформація про результати тестування

Для апаратної реалізації запропонованого методу використовується демонстраційний макет пристрою, який сигналізує про зміну температури та вологи у приміщенні за допомогою світлової індикації.

Даний макет реалізований за допомогою плати на базі мікроконтролера STM32F103C8T6 у ролі ПДТ. Роль ТП виконує відлагоджувальна плата Arduino UNO.

Передача даних між ПДТ і ТП відображена за допомогою програмного забезпечення логічного аналізатора Saleae Logic. Даний пристрій дозволяє відслідкувати зміну сигналів на портах введення-виведення мікроконтролера та за допомогою однойменного ПЗ відтворити ці дані на моніторі ПК (рис.4.7).

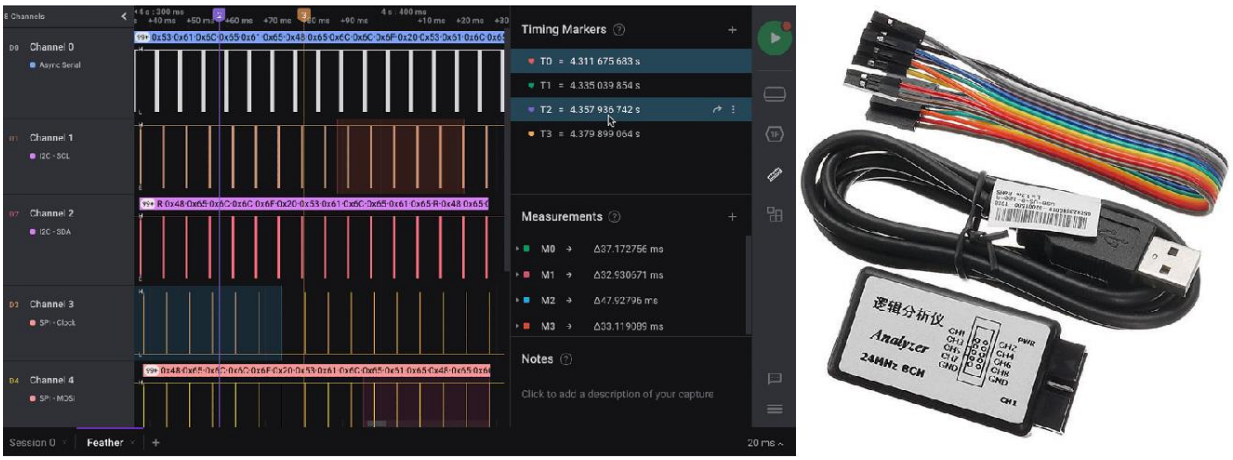


Рисунок 4.7 – Логічний аналізатор та ПЗ Saleae Logic

За реалізацією ПДТ та ТП інформація між ними передається за допомогою виділених GPIO. Фізичне з'єднання відбувається шляхом підключення дротів. Результат роботи буде відображений за допомогою зчитування сигналів на даних пінах під час тестування. Сигналом передачі даних від ПДТ до ТП є лінія CES (Command Execution Signal) При моделюванні даний сигнал буде постійно увімкнений і може виконувати роль підключення плати до живлення та демонстрації цього білим світлодіодом. Сигналом ініціації передачі даних, тобто сигналом від ТП до ПДТ є лінія IES (Initiate to Execute Signal).

Зчитування сигналів логічним аналізатором відбувається у односторонньому порядку. Достатньо підключити його до ТП, щоб повністю вивести результат роботи взаємодії ПДТ і ТП, так як лінія CES також може бути отримана від плати, що тестується.

Для повного аналізу та демонстрації результатів тестування усіх запропонованих сценаріїв, необхідно підключити логічний аналізатор до всіх важливих периферійних або конструктивних особливостей пристрою, що тестується (ТП).

Алгоритм коректної роботи макету у текстовому форматі наведений нижче відповідає позитивному тестовому сценарію від замовника. Також даний алгоритм відображено при зчитуванні даних логічним аналізатором.

Перед початком виконання, введемо умовні позначення ліній та сигналів. Сигнал підключення живлення ТП, увімкнення світлодіоду білого кольору, успішне виконання етапів тестування та запис показників у постійну пам’ять – White LED.

Результат вимірювання показників периферією (датчиком вологості та температури), негативний результат перевірки підключення даної периферії та подальші некоректні вимірювання – Red LED, Green LED, Blue LED.

1. Про підключення пристрою до живлення, сигналізує увімкнення світлодіоду білого кольору. Даний світлодіод двічі миготить. Логічний аналізатор відображає даний крок сигналом White LED (рис.4.8).

2. Відбувається перевірка підключення та коректність роботи периферії. Периферія в демонстраційному макеті – це датчик вологості, температури та три світлодіоди синього, зеленого та червоного кольорів. Логічний аналізатор відображає даний крок короткимсигналом White LED (рис.4.9).

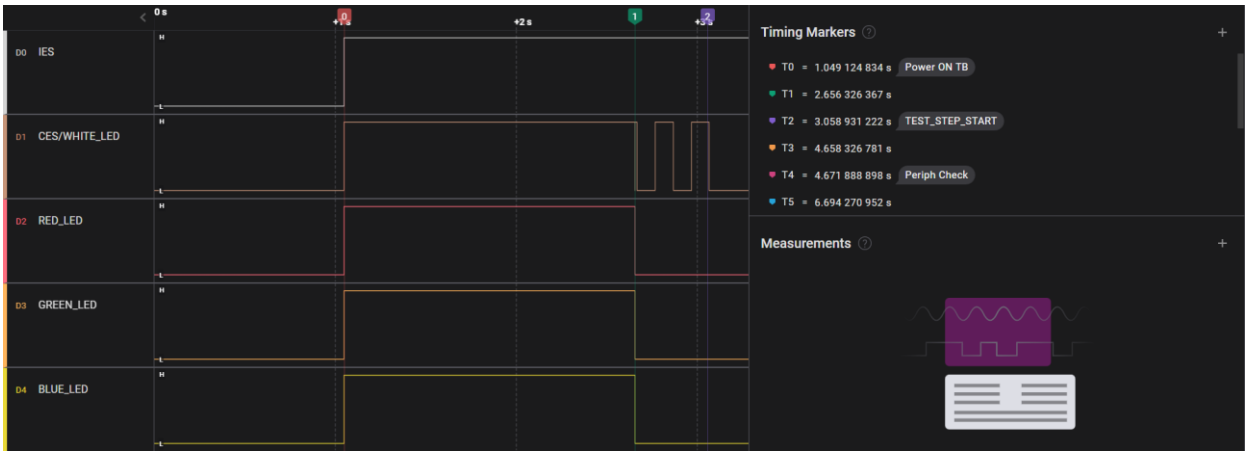


Рисунок 4.8 – Підключення живлення до пристрою та успішне сигналізування про підключення

Рисунок 4.9 – Крок перевірки периферії

3. Починається штатний режим роботи. Кожний певний проміжок часу, що встановлюється програмно, відбувається вимірювання температури та вологості. Якщо результати вимірювання знаходяться в межах норми, то про це сигналізує зелений світлодіод. Якщо результати нижче норми, то сигналізує синій, а вище норми – червоний. Виміри проводяться ту кількість разів, яка зазначена у сценарію від замовника. Даний етап тестування є великим по кількості даних та часу, але логічний аналізатор дозволяє це продемонструвати усього чотирьма лініями(сигналами) – White LED, Red LED, Green LED, Blue LED (рис.4.10).

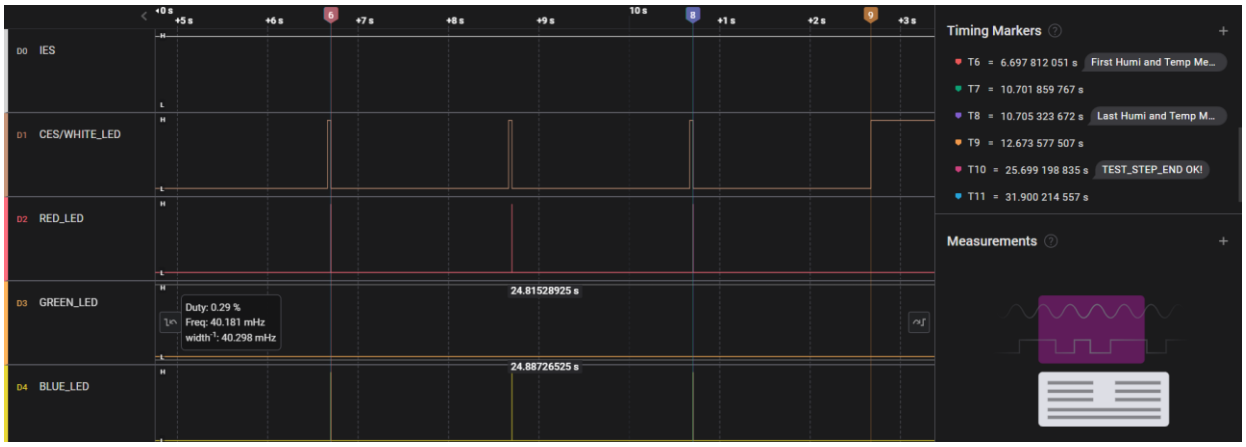


Рисунок 4.10 – Демонстрація вимірювання показників температуви та вологості.

4. При проходженні циклу вимірювання температури та вологості, дані записуються у постійну пам’ять пристрою, що тестується. На рис.4.10

зображено коректний результат запису даних у пам’ять за допомогою сигналу White LED.

5. Конструктивна особливість пристрою сигналізує про проходження етапів тестування та тесту в цілому змінним горінням білого світлодіоду кожну секунду. Інтервал горіння нескінченний із затримкою у 2 секунди (рис.4.11).

Рисунок 4.11 – Миготіння білого світлодіода як знак позитивного проходження тесту

6. Пристрій вимикається від живлення та вмикається через 10-12 секунд.

7. Після увімкнення, яке сигналізується білим світлодіодом, відбувається перевірка периферії (повторюється перший пункт алгоритму) для того, щоб запевнитися чи не постраждала вона після тестування. Периферія не постраждала, тестування пристрою вже відбулося –горить білий світлодіод. (рис.4.8, рис.4.9, рис.4.11).

Виникнення помилок у тестуванні, опис яких наведений нижче також відповідає чек-листу замовника.

В разі відсутності живлення пристрою, білий світлодіод не горить зовсім. Необхідно перевірити апаратну реалізацію пристрою, тобто його збірку. Даний крок можливо земулювати банальним відключенням дроту живлення ТП. Тоді сигнал White LED постійно буде мати рівень логічного

«0», як і всі інші сигнали ТП. (рис.4.12).

Рисунок 4.12 – Відсутність живлення ТП

На кроці перевірки периферії, якщо існує несправність, про це сигналізують постійним горінням синій (проблема з датчиком вологи) або червоний (проблема з ПЗП) світлодіоди. Якщо існує проблема з підключенням світлодіодів – горять всі три світлодіоди (синій, зелений та червоний; рис.4.13).

Рисунок 4.13 – Відсутність підключення будь-якого з трьох світлодіодів

На кроці штатного режиму роботи, якщо сенсори працюють некоректно (в основному це проблема з підключенням до мікроконтролера та коректністю результатів вимірювання показників вологості та температури), то всі три світлодіоди миготять певну кількість разів з затримкою після

кожного циклу миготіння. Цикл миготіння продовжується нескінченно до відключення пристрою. Кількість разів миготіння уніфікована переліком помилок, які можуть виникнути при тестуванні цифрового пристрою. Датчик вологості має циклічне миготіння 5 разів. Дані миготіння відображаються кількістю логічних «1» сигналів Red LED, Green LED та Blue LED при зчитуванні даних логічним аналізатором (рис.4.14).

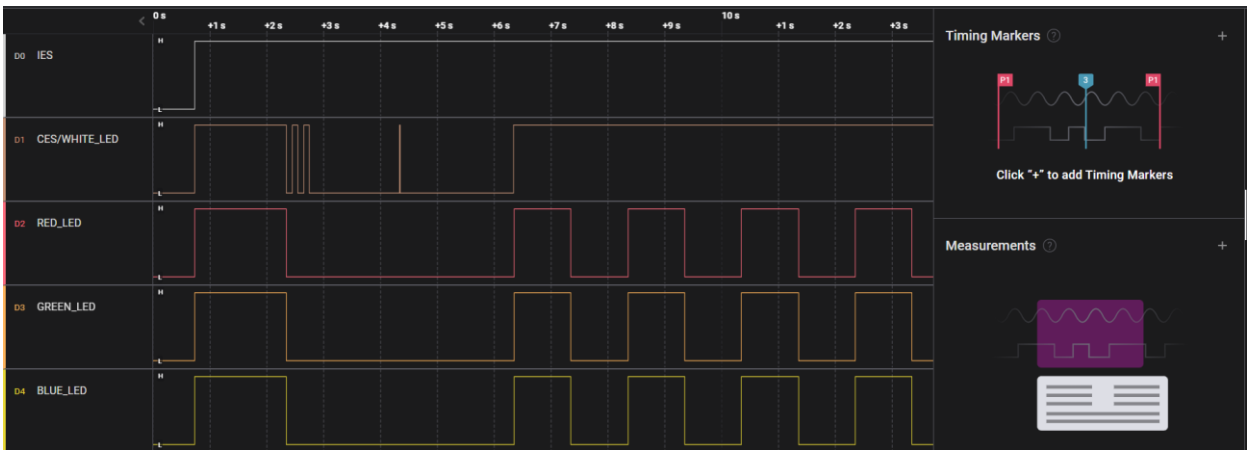


Рисунок 4.14 – Моделювання негативного тестування помилки читання даних з датчиків.

На кроці перевірки чи записалися результати тестування, алгоритм пошуку несправності периферії та даних тестування повторюється. В разі виходу з ладу периферії або відсутності результатів вимірювання (тестування), будуть горіти три світлодіоди (синій, зелений та червоний) (рис.4.13).

За алгоритмом роботи пристрою формується висновок, що ключовим фактором, насамперед, є перевірка коректності роботи периферії та всіх підключених компонентів виведення інформації за допомогою ПЗ. Але непрямим чином даний спосіб дозволяє вказати виробнику на проблеми апаратної та схемної реалізації пристрою (пошкодження мікроконтролера, підключення периферії, електронних компонентів тощо).

Testing Results

Executor : Engineer Name Surname

Approver: Manager of Manufacturing Name Surname
Customer Name Surname

Software version: TB_1_0_positive.bin
Testing scenario: positive
Count of test steps: 4/4
Successfully passed: 100%
Errors: 0

Software version: TB_1_0_negative.bin
Testing scenario: negative
Count of test steps: 4/4
Successfully passed: 100%
Errors: 4/4

Additional comments: None. Working well in both of scenarios.

Рисунок 4.16 – Приклад звіту тестування ТП з ПДТ

ВИСНОВКИ

В ході атестаційної роботи розглянуті методи діагностування цифрових пристроїв на різних технологічних платформах та сформульована постановка завдання на проектування системи діагностування критичного функціоналу пристрою із записом результатів.

Метою використання даного методу є його переваги, а саме:

– відсутність у потребі передавати оригінальне ПЗ, якщо перевірка пристроїв відбувається на виробництві у інших компаніях посередниках;

– відсутність потреби у постійній передачі готових пристроїв від замовника до виробника і навпаки на основі усних підтверджень чи пройшов пристрій тестування;

– запис результатів у постійну пам’ять для подальшого розшифрування чи дійсно пристрій був повністю протестований або на кому етапі тестування пристрій видав помилку;

– формування повідомлення про результати тестування у текстовому форматі та можливість відлагодити їх або відобразити на серверній частині при клієнт-серверній архітектурі;

– універсальність тестування для пристроїв однакових категорій (майже кожен сенсор тестується за одним і тим же сценарієм).

Недоліком є те, що тестування не гарантує точного визначення природи помилки при тестуванні (ПЗ чи помилка на етапі виробництва).

Виконано наступний алгоритм тестування.

1. Замовник надсилає спеціальне ПЗ для тестування з документацією до виробника. Документація має покроковий опис тестування та опис візуального відображення помилок чи позитивного результату, якщо дана можливість обумовлена конструкцією (наявність світлових або звукових індикаторів). Також в документації може бути вказано спосіб підключення пристрою до мережі Інтернет або до комп’ютера з віддаленим доступом.

2. Виробник тестує партію розроблених пристроїв.

3. Надсилає фото, відео або класичний тест-звіт за результатами тестування.

4. Замовник перевіряє запис результатів тестування під час відлагодження пристрою, який вже протестований чи за допомогою мережі Інтернет, якщо дозволяє обраний мікроконтролер або конструктивні особливості розробленого пристрою.

Апаратна реалізація методу діагностування цифрових пристроїв на різних технологічних платформах складається з двох плат, які узгоджено правилами підключення без відхилення від коректності роботи. Платою для тестування обрана плата на базі мікроконтролера STM32F103C8T6. Платою, що тестується обрана плата Arduino UNO. Виконуються правила узгодження логічних рівнів (3.3В ПДТ та 5В ТП), частот мікроконтролерів (частота ПДТ скорочена за допомогою середовища CubeMX), периферії (кількість пінів ПДТ та ТП достатня для проведення тестування) та передачі даних (ПДТ та ТП «спілкуються» між собою фізичного з'єднання двох пінів введення/виведення за схемою Output STM – Input Arduino та Input STM – Output Arduino).

Програмна реалізація ПДТ виконана за допомогою середовища CubeMX та Keil for ARM. Програмна реалізація ТП виконана за допомогою середовища Arduino IDE.

Результат програмно-апаратної реалізації зображений за допомогою зчитування сигналів логічним аналізатором Saleae Logic від ТП. Моделювання роботи системи проведено за «позитивним» та «помилковим» тестовим сценарієм.

Результати тестування надсилаються від Виробника до Замовника у вигляді класичного тест-звіту з переліком кількості проведених тест сценарієв, діагностованих помилок (вимушених за сценарієм або випадкових) та вказанням початку та закінчення часу тестування з відповідальною особою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Старченко Г. В. Управління проектами: теорія та практика [Текст] : навч. посіб. / Г. В. Старченко. – Чернігів : видавець Брагинець О. В., 2018. – 306 с.

2. Богданов, В.В. Управление проектами в Microsoft Project 2007 [Текст] : учеб. курс / Вадим Богданов. – СПб.: Питер Пресс, 2007. – 592 с.

3. Методы проектирования и поддержки требований к программному обеспечению [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Самар. гос. аэрокосм. ун-т им. С. П. Королева (СГАУ) ; [авт.-сост. А. В. Иващенко]. – Самара, 2013.

4. Lam W. K. Hardware Design Verification: Simulation and Formal Method-Based Approaches. – Prentice Hall PTR, 2005. – 624 p.

5. 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.

6. Broekman B. Testing Embedded Software / Bart Broekman, Edwin Notenboom. – Harlow, UK: Addison-Wesley, 2003. – 368 p.

7. Плати Ардуіно [Електронний ресурс] / Arduino.ua – Режим доступу: www / URL: https://doc.arduino.ua/ru/hardware/. – 10.12.2022 р. – Загол. з екрану.

8. Microcontrollers & Microprocessors [Електронний ресурс] / ST life.argumented – Режим доступу: www / URL: https://www.st.com/en/microcontrollers-microprocessors.html. – 10.12.2022 р. – Загол. з екрану.

9. Van der Hoeven J. Emulation for Digital Preservation in Practice: The Results / J. Van der Hoeven, B. Lohman, R. Verdegem // International Journal of Digital Curation. – 2007. – vol. 2. – no. 2. – P. 123–132