МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кваліфікаційна наукова
праця на правах рукопису

Чопей Ратібор Степанович

УДК 004.054

ДИСЕРТАЦІЯ

ЗАСОБИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ СПЕЦІАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВБУДОВАНИХ СИСТЕМ

01.05.03 – математичне і програмне забезпечення
обчислювальних машин і систем
(шифр і назва спеціальності)

05 "Технічні науки"
(галузь знань)

Подається на здобуття наукового ступеня
кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ Р.С. Чопей

(підпис, ініціали та прізвище здобувача)

Науковий керівник –
Федасюк Дмитро Васильович,
д.т.н., професор

Ідентичність всіх примірників дисертації
ЗАСВІДЧУЮ:
Вчений секретар спеціалізованої
вченої ради

/Р. А. Бунь/

Львів – 2018

АНОТАЦІЯ

Чопей Р. С. Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 01.05.03 – математичне і програмне забезпечення обчислювальних машин і систем (05 – Технічні науки). – Національний університет "Львівська політехніка" МОН України, Львів, 2018.

В дисертаційній роботі розв’язано актуальне наукове завдання удосконалення наявних методів тестування тривалості виконання програмного коду вбудованих систем реального часу і створення відповідного програмного засобу для автоматизації процесу їх тестування.

У вступі обґрунтовано актуальність теми дисертаційної роботи, сформульовано мету та основні задачі досліджень, визначено наукову новизну роботи і практичне значення отриманих результатів, показано зв'язок роботи з науковими темами. Подано відомості про апробацію результатів роботи, особистий внесок автора та його публікації.

У першому розділі дисертаційної роботи проведено аналіз сучасного стану розроблення вбудованих систем, в межах якого визначено особливості розробки та тестування його програмного забезпечення. Проаналізовано сучасний стан методів, моделей, засобів для аналізу тривалості виконання програмного коду (ТВПК). Проведений огляд наявних методів аналізу ТВПК дає змогу стверджувати, що здебільшого з них не повною мірою реалізована потрібна функціональність (врахування поведінки периферійних пристроїв, а також впливу на них зовнішніх і внутрішніх чинників), недостатньо висока точність розрахунку тривалості виконання програмного коду мікроконтролерних вбудованих систем. Тому важливим завданням дисертаційної роботи є розроблення методів аналізу ТВПК мікроконтролерних вбудованих систем реального часу з високим рівнем їх автоматизації.

У другому розділі сформульовано принципи, які потрібно враховувати під час розроблення методів і засобів тестування тривалості виконання програмного коду мікроконтролерних вбудованих систем. На їх основі запропоновано динамічні методи аналізу тривалості виконання програмного коду, що забезпечують можливість вимірювання тривалості виконання програмного коду безпосередньо на вбудованій системі без внесення змін у її програмне чи апаратне забезпечення. В основу метода покладено ідею вимірювання тривалості перебування між двома точками зупинки (breakpoint), що встановлені на початку та наприкінці фрагменту програми, тривалість якої потрібно виміряти.

Розроблено алгоритми та блок-схеми програмних модулів, що реалізують запропоновані динамічні методи аналізу тривалості виконання програмного коду, проведено верифікацію запропонованих методів.

У третьому розділі розроблено модель функціонування програмного забезпечення, що ґрунтується на поведінці периферійних пристроїв. Ця модель покладена в основу метода формування плану тестування, в якому передбачено проведення дослідження впливу зміни часу відгуку периферійних пристроїв на ТВПК. Для цього застосовано метод Монте-Карло, що дає змогу оцінити мінімальне та максимальне значення випадкової складової часу виконання функції. Поєднання отриманих результатів з інформацією про вагу потоку, у якому викликається функція, дає змогу розрахувати вагу функції та на підставі цих даних сформувати план тестування.

Здійснено розрахунок обчислювальної складності методу формування плану тестування, згідно з яким встановлено придатність запропонованого методу для великих проектів вбудованих систем.

У четвертому розділі подано аналіз впливу зовнішніх і внутрішніх чинників на ТВПК. Запропоновано метод аналізу ТВПК, що враховує вплив зовнішніх та внутрішніх чинників на компоненти вбудованої системи.

Результати дослідження ефективності застосування розробленого методу показують, що старіння компонентів найбільше впливають на ТВПК. Саме

тому проведено дослідження можливості використання цього методу для прогнозування тривалості виконання програмного коду в заданий момент часу. Отримані результати підтвердили потребу проведення прогнозування тривалості виконання програмного коду для визначення максимального часу експлуатації вбудованої системи, при якому вона не перевищує її крайній термін, а також для визначення періоду технічного обслуговування (ТО), що забезпечує її заданий рівень впродовж усього терміну експлуатації.

У п’ятому розділі подано опис розробленого програмного засобу для автоматизованого тестування ТВПК вбудованих систем жорсткого реального часу. У розробленому програмному засобі реалізовано такі основні можливості:

1) синтаксичний аналіз тексту програми;
2) автоматизоване тестування тривалості виконання програмного коду;
3) автоматизоване генероування плану тестування з урахуванням та без урахування поведінки периферійних пристроїв;
4) прогнозування тривалості виконання програмного коду з урахуванням зовнішніх і внутрішніх чинників;
5) візуалізація та збереження результатів (план тестування, результати тестування, результати прогнозування).

Для розроблення програмного засобу було використано такі технології: мову програмування C#, платформу .NET, технологію WPF для реалізації графічного інтерфейсу, бібліотеку µVision Socket Interface для доступу до функцій середовища розробки Keil µVision з сторонніх програмних модулів, а також бібліотеки iText/iTextSharp для формування звітів у форматі PDF.

Автоматизований аналіз тривалості виконання програмного коду вбудованих систем з урахуванням поведінки периферійних пристроїв, а також впливу на них зовнішніх та внутрішніх чинників, зробило можливим зростання точності отриманих результатів, та їх достовірності і водночас дало змогу зменшити витрати часу на тестування вбудованої системи загалом.

Ключові слова: тестування програмного забезпечення, тестування вбудованих систем, вбудовані системи реального часу, тривалість виконання програмного коду.

Список публікацій здобувача:

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Fedasyuk D. A method of predicting the maintenance period of embedded systems for preventing breach of their time requirements / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Computing, Ukraine. 2018. – Vol. 17. – Issue 2, pp. 94–101. (Scopus; Google Scholar; Index Copernicus).

2. Fedasyuk D. A Model for Estimating Firmware Execution Time Taking Into Account Peripheral Behavior / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Intelligent Systems and Applications, Hong Kong. – 2018. – Vol. 10. – No. 6, pp 22-29. (Scopus; Google Scholar; Microsoft Academic Search; Stanford University Libraries).

3. Fedasyuk D. Determining the Influence of the External Factors on the Firmware Response Time / D. Fedasyuk, V. Gavrysh, T. Marusenkova, R. Chopey // Central European Researchers Journal, Slovakia. – 2018. – Vol. 4. – Issue 1, pp. 1–8.

4. Федасюк Д. В. Алгоритм побудови графу потоку керування за текстом програми мовою С / Д. В. Федасюк, Р. С. Чопей // Науковий журнал Радіоелектроніка, інформатика, управління. – Запоріжжя: Запорізький національний технічний університет, 2018. № 2. – С. 154-161. (Thomson Reuters Web of Science; Google Scholar; eLIBRARY.RU; Index Copernicus).

5. Чопей Р. С. Огляд підходів до аналізу тривалості виконання програмного коду / Р. С. Чопей, Д. В. Федасюк. // Науково-технічний журнал "Електротехнічні та Комп’ютерні системи". – Одеса: Одеський національний політехнічний університет, 2017. – №26 (102). – С. 68-78. (eLIBRARY.RU; Index Copernicus).

6. Чопей Р. С. Метод аналізу тривалості виконання програмного коду з урахуванням архітектури мікроконтролера / Р. С. Чопей, Д. В. Федасюк //

Вісник Вінницького політехнічного інституту – Вінниця, 2018, №2(137). – С. 74-79. (Index Copernicus; РИНЦ).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

7. Fedasyuk, D. Architecture of a Tool for Automated Testing the Worst-Case Execution Time of Real-Time Embedded Systems' Firmware. / D. Fedasyuk, R. Chopey, B. Knysh // The experience of designing and application of CAD systems in microelectronics of the XIVth International conference CADSM'2017, February 21-25. – Lviv, 2017. – P. 278-282. (SCOPUS).

8. Chopey, R. A Model For Estimating Firmware Execution Time Based On Peripherals' Time Behavior. / R. Chopey, D. Fedasyuk, T. Marusenkova // Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering of the XIVth IEEE International Conference TCSET'2018, February 20-24. – Lviv, 2018. – P. 1179-1183. (SCOPUS).

9. Chopey R. The model of software execution time remote testing. / R. Chopey, B. Knysh, D. Fedasyuk // Computer Science and Engineering 2017 of the IXth International Conference of Young Scientists CSE'2017, November 22-24. – Lviv, 2017. – P. 398-402.

10. Чопей, Р. С. Метод побудови засобу для автоматизованого тестування тривалості виконання програмного коду вбудованих систем розроблених з використанням KEIL μVISION / Р. С. Чопей, Д. В. Федасюк // Електротехнічні та Комп'ютерні Системи: Теорія та Практика (ЕЛТЕКС – 2018) : матер. Міжнар. наук.-практ. конф., 29 травня – 1 червня 2018, Одеса, 2018. – С. 34-40.

11. Чопей, Р.С. Проблеми тестування вбудованих систем жорсткого реального часу / Р. С. Чопей, Д. В. Федасюк // Математичне і програмне забезпечення інтелектуальних систем (MPZIS-2016) : матер. XIV Міжнар. наук.-практ. конф., 16-18 листопада 2016, Дніпро. – Дніпро, 2016. – С. 243-245.

12. Чопей, Р.С. Тестування тривалості виконання програмного коду мікроконтролерних систем реального часу / Р. С. Чопей, Д. В. Федасюк // Математичне і програмне забезпечення інтелектуальних систем (MPZIS-2017) :

матер. XV Міжнар. наук.-практ. конф., 22-24 листопада 2017, Дніпро. – Дніпро, 2017. – С. 216-218.

ABSTRACT

Chopey R. S. Means of automated testing of specialized software of embedded systems. – Qualification scientific paper, manuscript.

Thesis for a Candidate Degree in Technical Sciences on specialty 01.05.03 – «Mathematical and software support of computer machines and systems». - Lviv Polytechnic National University, the Ministry of Education and Science of Ukraine, Lviv, 2018.

This dissertation presents a solution for an actual scientific problem of improving the existing methods of execution time analysis for real-time embedded systems and the corresponding software for automation the process of their testing.

The first chapter contains an analysis of the current state of the development of embedded systems within which specific features of the development and testing of its software have been determined. Established the current state of methods, models, means for execution time analysis. The overview of existing methods of firmware execution time analysis suggests that most of them do not fully implement the necessary functionality (take into account the behavior of peripheral devices, as well as the influence of external and internal factors on them), therefore, have insufficiently high accuracy of the calculation of the firmware execution time. Therefore, development of methods for execution time analysis for embedded real-time systems with a high degree of formalization, for its automation is important.

The second chapter presents the principles that need to be taken into account when developing methods and tools for testing the firmware execution time of the microcontroller embedded systems. On their basis dynamic methods of analysis of the duration of implementation of the code are offered, which provide an opportunity to measure the code execution time directly on the embedded system without modifying its software or hardware. The method is based on the idea of measuring the length of stay between two breakpoints set at the beginning and at the end of the program fragment, the execution time of which must be measured.

The algorithms and architecture of software modules that implement the proposed dynamic methods for analyzing the duration of the implementation of the code are developed, the verification of the proposed methods has been carried out.

The third chapter presents the developed functional model that based on the behavior of peripheral devices. This model is the basis of the method of forming a testing plan, which provides for the study of the influence of changing the response time of peripheral devices on the duration of the implementation of the code. For this purpose, the Monte Carlo method was used, which allows to estimate the minimum and maximum values of the random component of the function execution time. The combination of the obtained results with the information on the weight of the thread in which the function is called, allows you to calculate the weight of the function and, based on these data, generate a testing plan.

Calculated the computational complexity of the method of formation of the test plan, according to which the suitability of the proposed method for large projects of embedded systems is established.

The fourth chapter presents an analysis of the influence of external and internal factors on the firmware execution time. Presented method of firmware execution time analysis, which considers the influence of external and internal factors on the components of the embedded system.

The results of the evaluating of the effectiveness of the application of the developed method show that the aging of the components most affected the firmware execution time. That is why evaluated the possibility of using this method to predict the software execution time. The obtained results confirmed the need for prediction the firmware execution time for determine the maximum operating time of the embedded system, in which it does not exceed its deadline, as well as to determine the period of maintenance, which will ensure its specified level throughout the life of the operation.

The fifth chapter contains the description of developed software tool for automated testing the software execution time of embedded hard real-time systems. The developed software has the following key features:

1) syntactical analysis of the text of the program;
2) automated testing of the software execution time;
3) automated generating the testing plan that taking into account behavior of peripheral devices;
4) prediction the software execution time considering external and internal factors;
5) visualization and saving the results (test plan, test results, forecasting results).

Software tool was developed using the following technologies: the C # programming language, the .NET platform, the WPF for implementing the graphical interface, the μVision Socket Interface library for accessing Keil μVision development features from third-party software modules, and the iText / iTextSharp libraries for reporting purposes in PDF format.

Automated analysis of the embedded systems software execution time, taking into account the behavior of peripheral devices, as well as the influence of external and internal factors, has made it possible to increase the accuracy of the results obtained and their reliability, while also reducing the time spent on testing the embedded system in general.

Keywords: software testing, testing of embedded systems, real-time embedded systems, execution time.

The list of author's publications:

Proceedings where basic scientific results of thesis were published:

1 Fedasyuk D. A method of predicting the maintenance period of embedded systems for preventing breach of their time requirements / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Computing, Ukraine. 2018. – Vol. 17. – Issue 2, P. 94–101. (Scopus; Google Scholar; Index Copernicus).

2 Fedasyuk D. A Model for Estimating Firmware Execution Time Taking Into Account Peripheral Behavior / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Intelligent Systems and Applications, Hong Kong. – 2018. –

Vol. 10. – No. 6, P 22-29. (Scopus; Google Scholar; Microsoft Academic Search; Stanford University Libraries).

3 Fedasyuk D. Determining the Influence of the External Factors on the Firmware Response Time / D. Fedasyuk, V. Gavrysh, T. Marusenkova, R. Chopey // Central European Researchers Journal, Slovakia. – 2018. – Vol. 4. – Issue 1, P. 1–8.

4 Fedasyuk D. The algorithm of constructing control flow graph based on a program written in C / D. Fedasyuk, R. Chopey // The scientific journal Radio Electronics, Computer Science, Control, Zaporizhzhia. – 2018. – № 2. P. 154-161. (Thomson Reuters Web of Science; Google Scholar; eLIBRARY.RU; Index Copernicus).

5 Fedasyuk D. Overview of approaches to the execution time analysis of programs / D. Fedasyuk, R. Chopey // Scientific and Technical Journal Electrotechnic and Computer systems, Odessa. – 2017. – № 26 (102). P. 68-78. (eLIBRARY.RU; Index Copernicus).

6 Chopey R. Execution Time Analysis Method Taking Into Account Architecture of Microcontroller / R. Chopey, D. Fedasyuk// Visnyk of Vinnytsia Polytechnical Institute, Vinnytsia. 2018. – № 2 (137). P. 74-79. (Index Copernicus; РИНЦ).

Proceedings that certify an approvement of thesis materials:

7. Fedasyuk, D. Architecture of a Tool for Automated Testing the Worst-Case Execution Time of Real-Time Embedded Systems' Firmware. / D. Fedasyuk, R. Chopey, B. Knysh // The experience of designing and application of CAD systems in microelectronics of the XIVth International conference CADSM'2017, February 21-25. – Lviv, 2017. – P. 278-282. (SCOPUS).

8. Chopey, R. A Model For Estimating Firmware Execution Time Based On Peripherals' Time Behavior / R. Chopey, D. Fedasyuk, T. Marusenkova // Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering of the XIVth IEEE International Conference TCSET'2018, February 20-24. – Lviv, 2018. – P. 1179-1183. (SCOPUS).

9. Chopey R. The model of software execution time remote testing /

R. Chopey, B. Knysh, D. Fedasyuk // Computer Science and Engineering 2017 of the IXth International Conference of Young Scientists CSE'2017, November 22-24. – Lviv, 2017. – P. 398-402.

10. Chopey R. The method of construction of the means for execution time testing the embedded systems that are developed in keil μVision / R. Chopey, D. Fedasyuk // International conference "Electroengineering and Computer systems: Theory and Practice", Odesa, 29 of May – 1 of June 2018. P. 34-40.

11. Chopey R. The problems of testing the embedded hard real-time systems / R. Chopey, D. Fedasyuk // International conference "Mathematical and software of intelligent systems", Dnipro, November 16-18, 2016. P 243-245.

12. Chopey R. The problems of testing the embedded hard real-time systems / R. Chopey, D. Fedasyuk // International conference "Mathematical and software of intelligent systems", Dnipro, November 16-18, 2016. P 243-245.

13. Chopey R. Execution time testing of microcontroller based real time embedded system / R. Chopey, D. Fedasyuk // International conference "Mathematical and software of intelligent systems", Dnipro, November 22-24, 2017. P 216-218.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ЖЦ – життєвий цикл

ТО – технічне обслуговування

РЧ – реального часу

ТВПК – тривалість виконання програмного коду

SPI – Serial Peripheral Interface (послідовний периферійний інтерфейс)

I2C – Inter-Integrated Circuit (послідовна шина даних для зв'язку інтегральними мікросхемами)

UART – universal asynchronous receiver/transmitter (універсальний асинхронний приймач/передавач)

ВСТУП

Актуальність теми. Зростання продуктивності мікроконтролерів, що зумовлено застосуванням 32-бітних процесорів з архітектурою ARM спричинило розширення сфер застосування вбудованих систем, у тому числі уможливило їх використання для систем реального часу. Такі системи широко використовуються в різних галузях життєдіяльності людей (автомобільна, авіація, виробництво, медицина, сільське господарство, тощо).

Системи реального часу характеризуються тим, що в них правильність одержаних результатів залежить не тільки від логічної коректності програми, але й від проміжку часу, за який вони отримані. Залежно від наслідків, зумовлених порушенням часових вимог, системи реального часу прийнято поділяти на системи “м’якого” та “жорсткого” реального часу. У системі “м’якого” реального часу порушення часових вимог призведе до зниження якості роботи системи. Водночас, у системах “жорсткого” реального часу порушення часових вимог є рівнозначним відмові системи, а відтак, може, призвести до катастрофічних наслідків. Саме до таких систем висуваються більш жорсткі вимоги щодо їхньої надійності та безпечності. Забезпечення цих вимог є неможливим без застосування засобів автоматизованого тестування програмного коду. При цьому для систем жорсткого реального часу тестування тривалості виконання програмного коду набуває особливої актуальності.

Ще у 1990-х роках низка науковців (J. Engblom, A. Colin, G. Bernat, R. Chapman, C. Ferdinand, C. Healy, P. Atanassov та ін.) працювала над питаннями розроблення й дослідження моделей та методів аналізу тривалості виконання програмного коду вбудованих систем, що забезпечують низький рівень похибки отриманих результатів. У зв’язку із зростанням складності архітектури мікроконтролерів, а також ускладненням та розширенням функціоналу вбудованих систем зростає потреба в розробленні нових методів вимірювання тривалості виконання програмного коду, які не передбачають створення моделей апаратного забезпечення.

Одним із важливих упущень наявних методів є те, що жоден із них не дає

змогу враховувати вплив зовнішніх і внутрішніх чинників на вбудовану систему та зміну тривалості виконання програмного коду під дією цих впливів. Сукупність наведених обставин породила актуальне наукове завдання удосконалення наявних методів тестування тривалості виконання програмного коду вбудованих систем реального часу і створення відповідного програмного засобу для автоматизації процесу їх тестування. Актуальність цього наукового завдання, його практичне значення й зумовили вибір теми, мети, завдань і структури дисертаційного дослідження.

Зв'язок роботи з науковими програмами, планами, темами. Дисертація відповідає науковому напрямку кафедри програмного забезпечення Національного університету «Львівська політехніка» "Програмне і математичне забезпечення автоматизованих систем". Дисертація виконана в межах науково-дослідних робіт фінансованих Міністерством освіти і науки України та госпдоговірних науково-дослідних робіт, які виконувалися в Національному університеті «Львівська політехніка»: "Розроблення математичного забезпечення для програмного засобу аналізу функціональної безпечності та надійності програмно-апаратних систем відповідального призначення" (№ держреєстрації 0117U004458, 2018); госпдоговірні науково-дослідні роботи №0538, 0565 та 0581, що виконувалися впродовж 2016–2018 р. за тематикою "Автоматизоване тестування вбудованого програмного забезпечення". У перелічених науково-дослідних роботах автор розробив методи тестування тривалості виконання програмного коду вбудованих систем реального часу і створив відповідний програмний засіб для автоматизації процесу їх тестування.

У межах перелічених науково-дослідних робіт автор розробив математичне та програмне забезпечення для автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем.

Мета та завдання дослідження. Метою дисертаційної роботи є розроблення методів та алгоритмів аналізу тривалості виконання програмного коду вбудованих систем із урахуванням поведінки периферійних пристроїв, які

забезпечують заданий рівень надійності під час дії на них зовнішніх та внутрішніх чинників.

Для досягнення зазначеної мети поставлені і вирішені такі завдання:

1) Проаналізувати структурні та функціональні особливості процесу тестування мікроконтролерних вбудованих систем реального часу, що дасть змогу сформувати принципи, яких потрібно дотримуватися при розробленні програмного засобу для їх тестування.

2) Проаналізувати наявні методи й засоби аналізу тривалості виконання програмного коду, встановити їх переваги та недоліки, що дасть змогу визначити напрямки їх удосконалення для підвищення ефективності.

3) Удосконалити метод статичного аналізу тривалості виконання програмного коду для мікроконтролерних вбудованих систем, що враховує їхню внутрішню архітектуру, що дасть змогу врахувати втрати швидкості надсилання даних між внутрішніми модулями та при надсиланні даних через інтерфейси зв’язку.

4) Розвинути метод динамічного тестування тривалості виконання програмного коду вбудованих систем, який враховує особливості мікроконтролерів з архітектурою ARM, що дасть змогу використовувати його для більшості сучасних аналогічних вбудованих систем.

5) Розробити метод формування плану тестування програмних функцій вбудованої системи, який враховує поведінку периферійних пристроїв вбудованої системи, що забезпечить підвищення ефективності її тестування.

6) Розробити метод прогнозування тривалості виконання програмного коду, який ґрунтується на аналізі впливу зовнішніх та внутрішніх чинників на периферійні пристрої вбудованої системи, що дасть змогу підвищити точність отриманих результатів.

7) Розробити, реалізувати та верифікувати програмний засіб для тестування тривалості виконання програмного коду на підставі розроблених методів, моделей та алгоритмів, що дасть змогу автоматизувати процес тестування спеціалізованого ПЗ вбудованих систем реального часу.

Об'єктом дослідження є автоматизоване тестування тривалості виконання програмного коду вбудованих систем реального часу.

Предметом дослідження є методи та алгоритми тестування тривалості виконання програмного коду вбудованих систем реального часу з урахуванням поведінки периферійних пристроїв, а також впливу на них зовнішніх та внутрішніх чинників.

Методи дослідження. Для вирішення поставлених у дисертаційній роботі завдань використано такі методи дослідження: методи натурного експерименту та теорії алгоритмів – для методів динамічного аналізу тривалості виконання програмного коду; теорію ймовірності та математичну статистику – для побудови моделі функціонування програмного забезпечення, що враховують поведінку периферійних пристроїв; теорія системного аналізу та теорія математичного моделювання – для оцінювання впливу зовнішніх та внутрішніх чинників на ТВПК; теорію алгоритмів, принципи аспектно- та об'єктно-орієнтованого програмування – для розроблення програмних засобів.

Наукова новизна отриманих результатів. Основні результати роботи, які визначають її наукову новизну та виносяться на захист:

- *вперше розроблено* метод прогнозування тривалості виконання програмного коду, який враховує вплив зовнішніх і внутрішніх чинників на периферійні пристрої вбудованої системи РЧ, що дає змогу забезпечити заданий рівень їхньої надійності та безпечності.
- *удосконалено* метод статичного аналізу тривалості виконання програмного коду для мікроконтролерних вбудованих систем, який на відміну від наявних враховує швидкість обміну даними між внутрішніми модулями мікроконтролера, що дає змогу підвищити точність отриманих результатів.
- *отримали подальший розвиток:*
  - метод формування плану тестування програмних функцій вбудованої системи, в якому, на відміну від відомого, враховано поведінку її периферійних пристроїв, що дає змогу оцінити важливість тестування

кожної програмної функції, і в такий спосіб підвищити ефективність процесу автоматизованого тестування.

– метод динамічного тестування програмного забезпечення вбудованих систем з використанням інтерфейсу відлагодження, в якому, на відміну від відомого, враховано особливості мікроконтролерів з архітектурою ARM, що дає змогу виконувати тестування тривалості виконання програмних функцій вбудованої системи без внесення змін до її програмного чи апаратного забезпечення.

Практичне значення отриманих результатів.

1. Розроблено архітектуру програмного засобу для автоматизованого тестування тривалості виконання програмного коду, використання якої дає змогу підвищити ефективність процесу тестування програмного забезпечення вбудованих систем.
2. Розроблено програмне та інформаційне забезпечення програмного засобу для автоматизованого тестування програмного забезпечення вбудованих систем, яке дає змогу виконувати тестування та прогнозування тривалості виконання програмного коду з урахування впливу зовнішніх і внутрішніх чинників на вбудовану систему.

Розроблений програмний засіб EXTT на основі розроблених методів впроваджено в процес тестування вбудованих систем компанії Dinamica Generale S.p.A (м. Поджо Руско, Італія), що підтверджено відповідним актом. Розроблений метод формування плану тестування, що враховує поведінку периферійних пристроїв, впроваджено в практику тестування вбудованих систем компанії ПП НВПП “Спаринг-Віст Центр” (м. Львів). Також, результати дисертаційного дослідження використовуються у навчальному процесі Національного університету «Львівська політехніка» для дисциплін «Програмування мікроконтролерів». Використання та впровадження результатів дисертаційної роботи підтверджено відповідними актами.

Особистий внесок здобувача. Всі наукові результати дисертаційної роботи отримані здобувачем особисто. У друкованих працях, написаних у

співавторстві, здобувачеві належать: [1, 7, 10] – аналіз та формування напрямку наукових досліджень; [4] – метод аналізу тривалості виконання програмного коду з урахуванням архітектури мікроконтролера; [8, 9, 12] – архітектура засобу тестування тривалості виконання програмного коду та модель процесу віддаленого тестування; [2, 11] – модель для оцінювання тривалості виконання програмного коду з урахуванням поведінки периферійних пристроїв вбудованої системи; [5, 6] – метод урахування впливу зовнішніх та внутрішніх чинників на тривалість виконання програмного коду, та методу прогнозування ТВПК на підставі аналізу впливу зовнішніх та внутрішніх чинників.

Апробація результатів дисертації. Основні положення і результати роботи доповідалися та обговорювалися на 6 міжнародних наукових конференціях: XIV та XV Міжнародних науково-практичних конференціях “Математичне і програмне забезпечення інтелектуальних систем” «MPZIS» (Дніпро, 2016, 2017); XIV International Conference the Experience of Designing and Application of CAD Systems in Microelectronics «CADSM» (Поляна, 2017); IX International Academic Conference of Young Scientists «Computer Science and Engineering CSE-2017» (Львів, 2017); XIV International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering «TCSET-2018» (Славське, 2018); Міжнародній науково-практичній конференції «Електротехнічні та комп’ютерні системи: Теорія та практика ЕЛТЕКС-2018» (Одеса, 2018).

Результати дисертаційної роботи неодноразово обговорювалися на наукових семінарах кафедри програмного забезпечення Національного університету «Львівська політехніка».

Публікації. За результатами досліджень, які викладено в дисертаційній роботі, опубліковано 12 наукових праць, серед яких 3 [4, 5, 6] статті у наукових фахових виданнях України, 1 стаття у в науковому виданні України, що внесене до міжнародної наукометричної бази SCOPUS [1]; 2 статті [2, 3] у наукових періодичних виданнях інших держав; 6 публікацій у збірниках праць

міжнародних конференцій [7 – 12], серед яких 2 внесені в міжнародну наукометричну базу SCOPUS [7, 8].

Структура та обсяг роботи. Дисертація складається із вступу, п’яти розділів висновків, списку використаних джерел із 110 найменувань та 2 додатків. Загальний обсяг роботи становить 154 сторінок, з них 112 основного тексту, 45 рисунків та 35 таблиць.

РОЗДІЛ 1. СТАН ПРОБЛЕМИ ТЕСТУВАННЯ СПЕЦІАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВБУДОВАНИХ СИСТЕМ

Описано поняття вбудована система, галузі її застосування, а також особливості розроблення та тестування. Проведено огляд та порівняння статичних і динамічних методів аналізу тривалості виконання програмного коду, а також наявних програмних систем.

Обґрунтовано потребу створення нових моделей та засобів для тестування тривалості виконання програмного коду з допомогою апаратного трасування, яке дає змогу проводити тестування безпосередньо на вбудованій системі, без впливу на перебіг виконання програми, що забезпечує зростання ефективності процесу тестування, а також підвищує рівень їх надійність та безпечності.

За результатами проведеного аналізу стану проблеми визначено напрямки можливих досліджень.

1.1 Етапи розроблення вбудованих систем та галузі їх застосування

За останні десятиліття людство стало все більше залежним від систем, що вбудовані у різні пристрої. Саме тому поняття "вбудована система" набуло дуже багато означень. У наукових джерелах фігурує декілька визначень поняття "*вбудована система*", а саме:

– це поєднання апаратного і ПЗ, а також додаткових електронних або механічних частин, які потрібні для виконання обмеженої кількості функції [42].

– це спеціалізована комп'ютерна система або обчислювальний пристрій, призначений для виконання обмеженої кількості функцій, часто з обмеженнями РЧ, є комбінацією апаратного та ПЗ, з механічними або іншими частинами, що потрібні для виконання певної функції [59].

– це будь-яка комп’ютерна система окрім персонального комп’ютера [91].

Незалежно від обраного означення будь-яку вбудовану систему можна подати, як поєднання програмного та апаратного забезпечення (рис. 1.1). До складу апаратного забезпечення входить мікроконтролер (мікропроцесор), периферійні пристрої та різноманітні аналогові та цифрові сенсори. ПЗ містить вбудовану операційну систему та мікропрограму, що відповідає за виконання заданих функції.

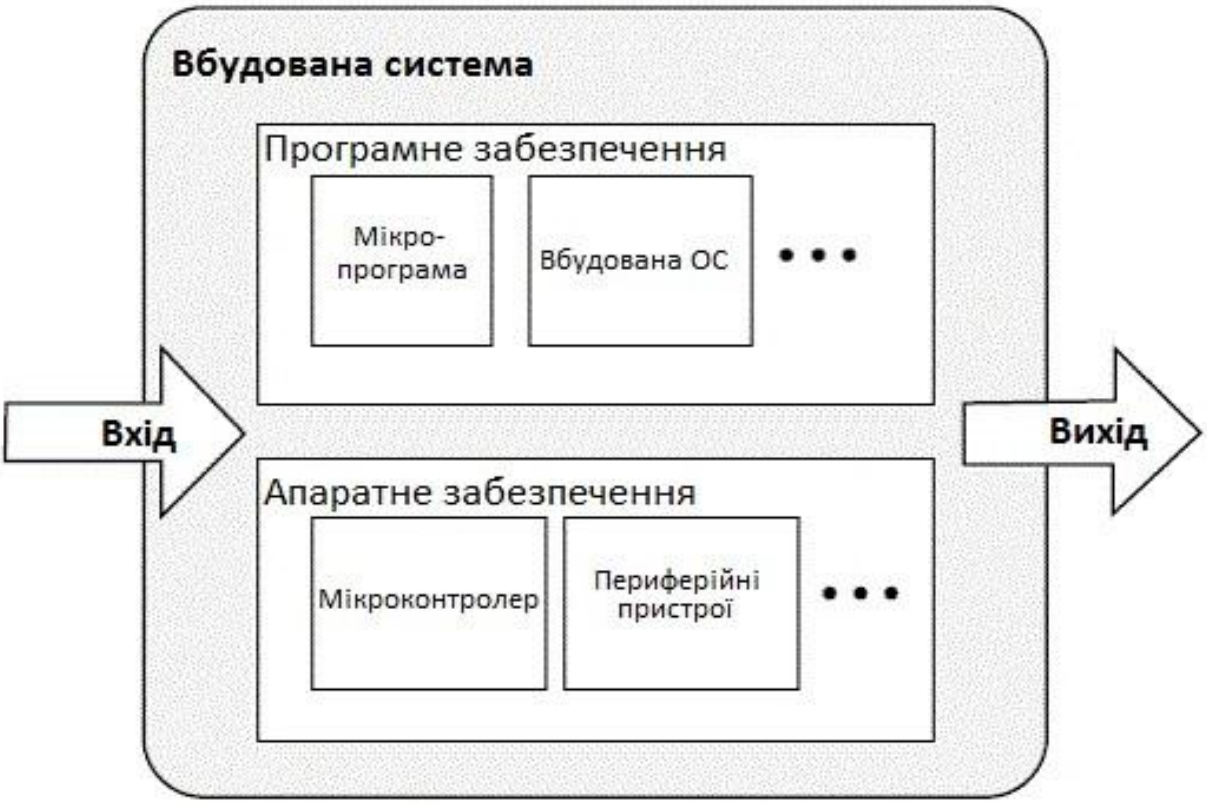


Рисунок 1.1 – Типова архітектура вбудованої системи

З огляду на те, що кожну вбудовану систему розробляють для виконання конкретного завдання чи завдань, тому їх життєвий цикл має типову структуру (рис. 1.2).

Формування вимог до вбудованої систем. Цей етап є основним під час проектування буд якої системи, а саме: визначають функції, які повинні виконувати вбудована система, наявні обмеження, вимоги до її надійності та безпечності, а також відповідність наявним міжнародним стандартам якості ПЗ.

Проектування архітектури вбудованої системи - передбачає формування вимог до її апаратного та програмного забезпечення. Неправильний розподіл

функцій вбудованої системи призведе до некоректного використання апаратних чи програмних ресурсів. Наприклад, виконання певної функції апаратними ресурсами, може, пройти швидше, ніж з допомогою програмного забезпечення. Також, на цьому етапі визначають можливі шляхи розвитку вбудованої системи та формують вимоги до апаратного забезпечення так, щоб вони відповідали сучасним доопрацюванням науково технічного прогресу, адже її апаратне забезпечення є незмінним впродовж усього періоду експлуатації, і не підлягає оновленню, як ПЗ.

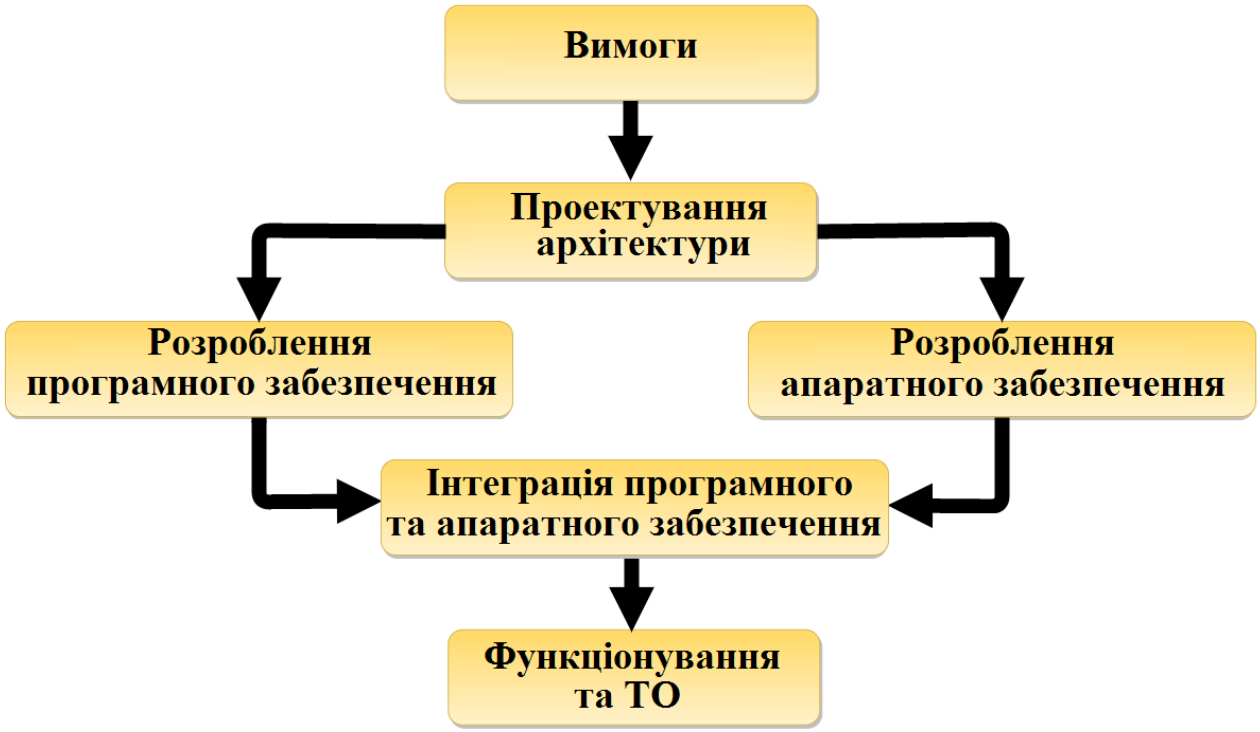


Рисунок 1.2 – Життєвий цикл вбудованої системи

Розроблення апаратного забезпечення вбудованої системи - передбачає виконання таких кроків: вибір процесора чи мікроконтролера, вибір периферійних пристроїв, вибір інтерфейсів зв'язку між периферійними пристроями, створення прототипів апаратного забезпечення, формування вимог до джерела живлення, а також побудова електричної принципової схеми та ескізу друкованої плати.

Розроблення програмного забезпечення вбудованої системи - виконуються шляхом вибору операційної системи, мови програмування, архітектури програмного забезпечення, бібліотек для

периферійних пристроїв, а також виконується побудова та реалізація відповідних алгоритмів.

Інтеграція програмного та апаратного забезпечення вбудованої системи та їх тестування - передбачає виконання таких дій: модульне тестування програмного забезпечення, інтегрування його на розроблену апаратну платформу (апаратне забезпечення), а також їх інтеграційне та системне тестування.

Функціонування та технічне обслуговування вбудованої - передбачає введення її в експлуатацію, оновлення її програмного забезпечення та здійснення технічного обслуговування, якщо це передбачено вимогами до неї.

Вбудовані системи характеризується низкою особливостей порівнюючи з системами, що розробляються на базі комп'ютерів, а саме:

- унікальністю апаратного забезпечення, яке розробляють винятково для виконання заданих функції;
- унікальністю програмного забезпечення, яке розробляється для наявного апаратного забезпечення;
- незмінністю до завершення терміну експлуатації, позаяк не можливо змінити функції вбудованої системи для яких вона розроблялась. Це зумовлено потребою змінити не тільки програмне, а й апаратне забезпечення;
- високою швидкодією, порівнюючи з персональними комп'ютерами через те, що вбудовані системи виконують обмежену кількість функцій, для яких їх розроблено. Це дає змогу значно підвищити не тільки їх швидкодію, але й продуктивність роботи загалом;
- апаратними обмеженнями, наприклад, мікроконтролери, що використовуються у вбудованих системах характеризуються порівняно невеликими обсягами пам'яті програм і пам'яті даних, а також не високою тактовою частотою;
- прогнозований час реакції на певні події, є основною вимогою для систем РЧ.

- низький рівень енергоспоживання, зумовлений тим, що апаратне забезпечення містить тільки ті компоненти, які потрібні для виконання визначених функції;
- малі габарити та маса, зумовлено потребою розміщення їх в середині інших систем;
- більш жорсткі вимоги до якості, надійності та безпечності, зумовлено постійним розширенням галузей їх застосування, у яких їх відмова призведе до катастрофічних наслідків.

Саме ці особливості вбудованих систем привели до їх популяризації та стрімкого розвитку, а відтак і до розширення галузей їх застосування. На сьогодні їх застосовують у різних галузях господарювання, а за даними маркетингових компаній [31, 32, 43] прогнозують, що до 2020 року ринок вбудованих систем зросте до 214,39 млрд. дол. США. Зокрема, очікується істотне зростання ринку у автомобільній та медичній галузях, а також у напрямку Інтернет-речей. Водночас, за даними цих досліджень, очікується, що питання надійності та безпеки вбудованих систем ймовірно стануть викликом для усіх їх розробників.

1.2 Потреба аналізу тривалості виконання програмного коду для вбудованих систем реального часу

Вбудовані системи РЧ є окремим підкласом вбудованих систем. Згідно з [73] в системах РЧ правильність одержаних результатів залежить не тільки від логічної коректності програми, але й від проміжку часу, за який їх отримано. Залежно від наслідків, спричинених порушеннями часових вимог, вбудовані системи РЧ прийнято поділяти на два типи [37]: системи жорсткого і м’якого РЧ.

У системах жорсткого РЧ порушення часових вимог є рівнозначним відмові системи, а відтак може призвести до катастрофічних наслідків, зокрема: заподіяння шкоди обладнанню, великій втраті доходів, або травмуванні чи загибелі людей. У системах м’якого РЧ порушення часових вимог призведе до тимчасового зниження якості роботи системи, що не супроводжуються

катастрофічними наслідками. На рис. 1.3 наведено приклади вбудованих систем і їх належність до певного класу.

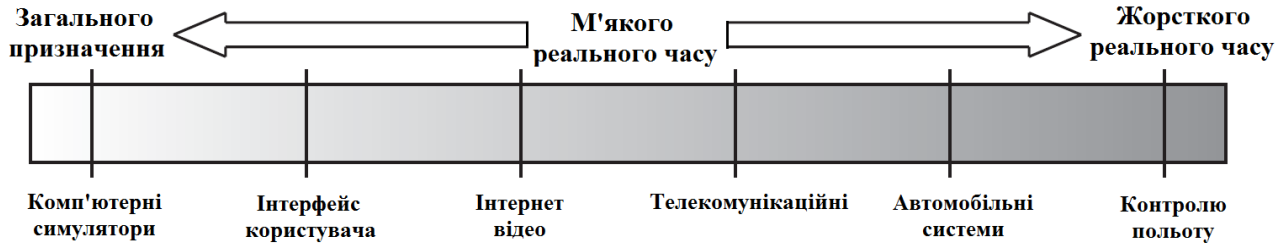


Рисунок 1.3 – Приклади систем реального часу відповідно до часових вимог виконання програми

Правильна часова поведінка є обов'язковою властивістю для будь-якої вбудованої системи жорсткого РЧ, яка має бути забезпечена впродовж усього терміну її експлуатації. Для дослідження часової поведінки використовуються аналіз тривалості виконання програмного коду під яким розуміють будь-який структурований метод чи інструмент, що застосовують для оцінювання тривалості виконання фрагменту програми. Найбільш важливим значенням такої тривалості є найбільша тривалість виконання програмного коду WCET (англ. *Worst-Case Execution Time*) під якою розуміють максимальний час, що потрібний для виконання, певного фрагменту коду в заданому контексті на заданому апаратному забезпеченні (рис. 1.4).

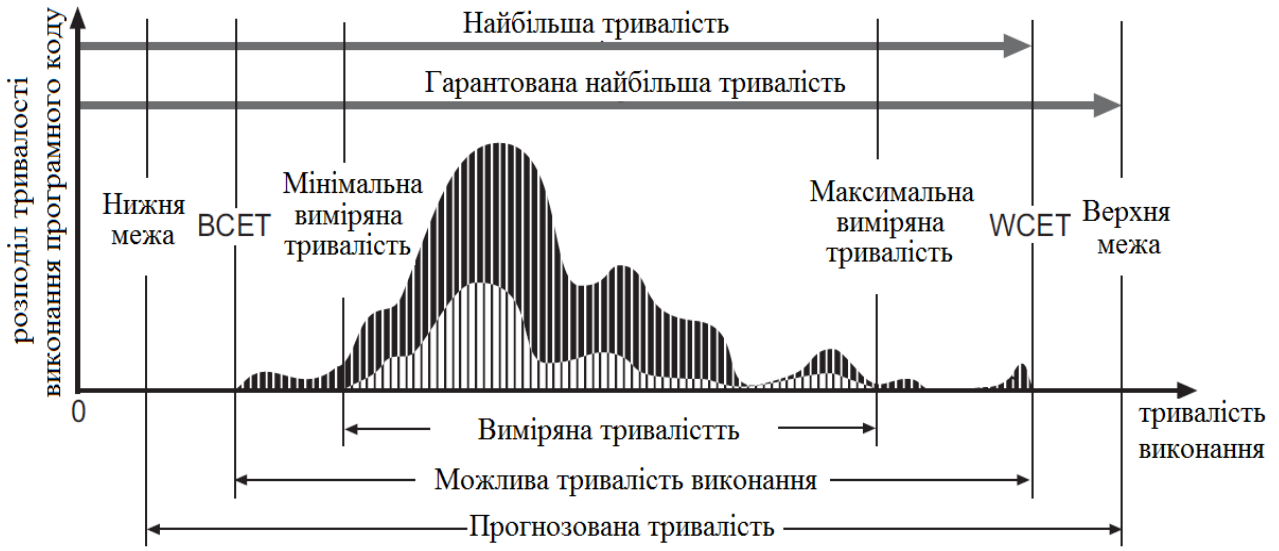


Рисунок 1.4 – Розподіл тривалості виконання програмного коду

Під час аналізу тривалості виконання програмного коду досліджують й інші показники, зокрема:

- *найменша тривалість виконання програми, BCET* (англ. *Best-Case Execution Time*) – це мінімальний час, що потрібний для виконання певного фрагменту коду в заданому контексті на заданому апаратному забезпеченні.
- *середня тривалість виконання програми, ACET* (англ. *Average-Case Execution Time*) – перебуває в межах між найбільшої та найменшої тривалістю виконання програми і залежить від відповідного розподілу часу.

Аналіз тривалості виконання програмного коду, а саме аналіз найбільшої тривалості, є дуже складним завданням, адже передбачає потребу проведення детального аналізу програмного забезпечення. Складність цього завдання збільшується з ростом простору можливих станів сучасного програмного забезпечення, а також з ростом складності архітектури процесорів, а саме: збільшення рівнів конвеєру команд, використання багаторівневої хеш-пам’яті.

Використання цього чи іншого методу аналізу тривалості виконання програмного коду, може, призвести до завищення (верхня межа) чи заниження (виміряних значень) результатів. Завищення отриманих результатів є більш безпечним ніж їх заниження, хоча і призводить до надмірного резервування програмних ресурсів вбудованої системи, а у деяких випадках і до зростання тривалості розроблення програмного забезпечення.

Заниження отриманих результатів є недопустимим під час аналізу показника найбільшої тривалості виконання програмного коду, однак є допустимим при аналізі показників найменшої та середньої тривалості виконання програмного коду.

1.3 Методи аналізу тривалості виконання програмного коду

Для оцінювання можливостей сучасних методів аналізу тривалості виконання програмного коду проведено детальне дослідження опублікованих

наукових робіт. Усі методи розділено на три групи, згідно з процесом їх виконання, а саме: статичні методи, динамічні методи та гібридні методи.

Поданий нижче огляд методів аналізу тривалості виконання програмного коду, що знайшли широке практичне застосування при тестуванні вбудованих систем, та їх коротка характеристика [97].

1.3.1 Статичні методи аналізу тривалості виконання програмного коду

Статичні методи аналізу тривалості виконання програмного коду є окремим класом методів, що не передбачають виконання програми на реальному обладнанні чи симуляторі. За вхідні дані вони використовують програмний код, а також модель апаратного забезпечення. На підставі аналізу можливих шляхів потоку виконання програми визначають її тривалість.

Традиційно статичні методи аналізу тривалості виконання програмного коду передбачають реалізацію таких етапів:

1) аналіз потоку виконання програми;
2) аналіз низькорівневих функцій;
3) розрахунок найбільшої тривалості виконання програмного коду.

Аналіз потоку виконання програми є основним етапом при статичному аналізі тривалості виконання програмного коду, який виконують шляхом синтаксичного аналізу тексту програми. Результатом виконання такого аналізу є інформація про усі функції програми, зокрема: можливі шляхи виконання програми, максимальну кількість ітерацій циклів, а також глибину рекурсії. До методів аналізу потоку виконання належать три методи: метод абстрактної інтерпретації [27], символьного виконання [69] та спеціалізований аналіз потоків даних. Загалом робота цих методі зводиться до синтаксичного аналізу тексту програми, вибір якого залежить від типу вхідних даних, наприклад, для аналізу тексту програми використовують підходи [30, 5], що побудовані на методі символьного виконання, для аналізу вихідного або проміжного коду використовують підходи [36, 44], що побудовані на методі абстрактної

інтерпретації, а для аналізу машинного коду застосовують підходи [78, 45], що використовують метод спеціалізованого аналізу потоків даних. Здебільшого методи синтаксичного аналізу тексту програми дають змогу тестувальнику вносити додаткову інформацію, яку подано як анотації до програмного коду, що дає йому змогу значно підвищити точність отриманих результатів, а відтак є передумовою для проведення безпечного оцінювання тривалості виконання програмного коду.

Аналіз низькорівневих функцій – передбачає визначення часової межі виконання функцій низького рівня (драйверів периферійних пристроїв). Для виконання такого аналізу потрібно врахувати архітектуру та особливості апаратного забезпечення вбудованої системи. Саме тому, для кожної з них потрібно створити відповідну модель, яка відтворює її поведінку при виконанні програми. Складність процедури створення моделей апаратного забезпечення вбудованих систем полягає у потребі побудови моделей не тільки мікроконтролера, а й усього апаратного забезпечення, що впливає на ТВПК. Окрім цього, побудова моделей апаратного забезпечення ускладнюється тим, що отримані дані про їх часові характеристики є наближеними, адже внутрішня архітектура мікроконтролера зберігається у секреті через жорстку конкуренцію між виробниками. Тому наведені у документації прості формули дають розробнику тільки наближене уявлення про часові характеристики мікроконтролерів, що призводить до спрощення їх часових моделей і, як наслідок, до зниження ступеня адекватності отриманих результатів.

В роботах [68, 20] запропоновано не створювати моделі апаратного забезпечення вбудованих систем. Відтак тривалість виконання певної функції визначають, як добуток кількості обчислювальних операцій потрібних для її виконання на тривалість виконання однієї обчислювальної операції. Цей підхід має низький рівень адекватності отриманих результатів, тому його не застосовують для оцінювання тривалості виконання програмного коду вбудованих систем жорсткого РЧ.

В роботах [1336, 14, 16, 46, 60] розглянуто вплив хешу на ТВПК. Такий аналіз має два напрями: аналіз хеш-пам’яті [6, 16], аналіз хешу інструкцій [14, 46, 52, 60]. У роботі [14] запропоновано модель статичного хешування, яка дає змогу аналізувати хеш-пам'ять що містить декілька рівнів. У роботі [41] запропоновано метод абстрактної інтерпретації для аналізу хешу інструкцій. В роботі [6] використано методі для аналізу хеш-пам’яті для визначення поведінки хешу у кожному вузлі синтаксичного дерева програми. У роботі [60] наведено підходи до введення обмежень, які потрібно застосовувати для аналізу хешу інструкцій та даних. Робота [57] поєднує результати аналізу хешу інструкцій та даних, які отримані при використанні симулятора процесора. Аналіз хешу, який використовує Stappert у роботі [74] пропонує метод аналізу потоку даних для визначення поведінки хешу даних для програм, що не містять циклів. Роботи Ferdinand та інших [67, 65] наводять результати аналізу функцій низького рівня для процесорів, що використовують єдиний хеш.

У роботах [23, 82] наведено методи, які дають змогу врахувати вплив архітектури конвеєра команд на ТВПК. Atanassov та інші [10] розробили модель конвеєра команд для процесора Infineon C167, що має константний час для виконання кожної програмної інструкції. У роботі [82, 1] для базової програми здійснено аналіз конвеєра команд у поєднанні з аналізом хешу інструкцій для процесора MIPS R3000/3010. У роботах [21, 24] наведено підхід та результати аналізу найбільшої тривалості виконання програмного коду для процесора Pentium. У роботі [74**Помилка! Джерело посилання не знайдено.**] запропонована спрощена модель конвеєра команда процесора PowerPC604, що містить також модель хеш-пам’яті та хешу інструкцій. У роботах [71, 63, 62] наведено результати аналізу конвеєра команд, що виконувався безпосередньо на апаратному забезпеченні, а саме – процесорі Pentium III та Athlon. Робота [58] стосується аналізу часових аномалій, що зумовлені хеш-пам’яттю.

У роботах [9, 3, 70] розглядають інші особливості, що впливають на ТВПК. Зокрема, в роботі [9] подано підхід, що враховує вплив динамічної оперативної пам’яті на ТВПК. У роботах [3, 70] наведено підходи, що дають

змогу врахувати затримки у багато-потокових системах та за рахунок цього підвищити точність отриманих результатів.

У роботі [70] звернуто увагу на можливість підвищення точності отримання результатів шляхом врахування втрати швидкості обміну даними між мікроконтролером і периферійними пристроями. Розрахунок втрат швидкості виконується за формулою:

$$V_{bcs} = \frac{f_{CPU}}{\text{НСД}(f_{CPU}, f_{BUS})}, \quad (1.1)$$

де: f_{CPU} – частота, з якою працює мікроконтролер; f_{BUS} – частота, з якою працює інтерфейс зв'язку; НСД() – найбільший спільний дільник. Експериментально доведено [54] підвищення точності отриманих результатів під час використання цього методу, однак практичне його застосування обмежене потребою його адаптації для конкретного мікроконтролера.

Розрахунок найбільшої тривалості виконання програмного коду виконується для об'єднання інформації, яка отримана на етапі аналізу потоку виконання програми та аналізу низькорівневих функцій. Тут застосовують один з трьох наявних методів розрахунку (рис. 1.5):

1) розрахунок на підставі дерева програми;
2) розрахунок на підставі шляху виконання програми;
3) розрахунок на підставі неявного перерахунку шляху.

Вибір методу розрахунку залежить від базового подання програми, а також характеристик потоку виконання програми.

Метод розрахунку на підставі дерева програми (*англ. tree-based WCET calculation)* є найбільш простим і широко відомим підходом. Його робота полягає у перетворенні тексту програми в орієнтоване дерево, у якому вузли описують структуру програми (послідовності, цикли або умовні об'єкти), а листками дерева є основні блоки. Процес розрахунку найбільшої тривалості виконання програми передбачає трансформацію графу потоку управління згідно з правилами, наведеними на рис. 1.5 г.

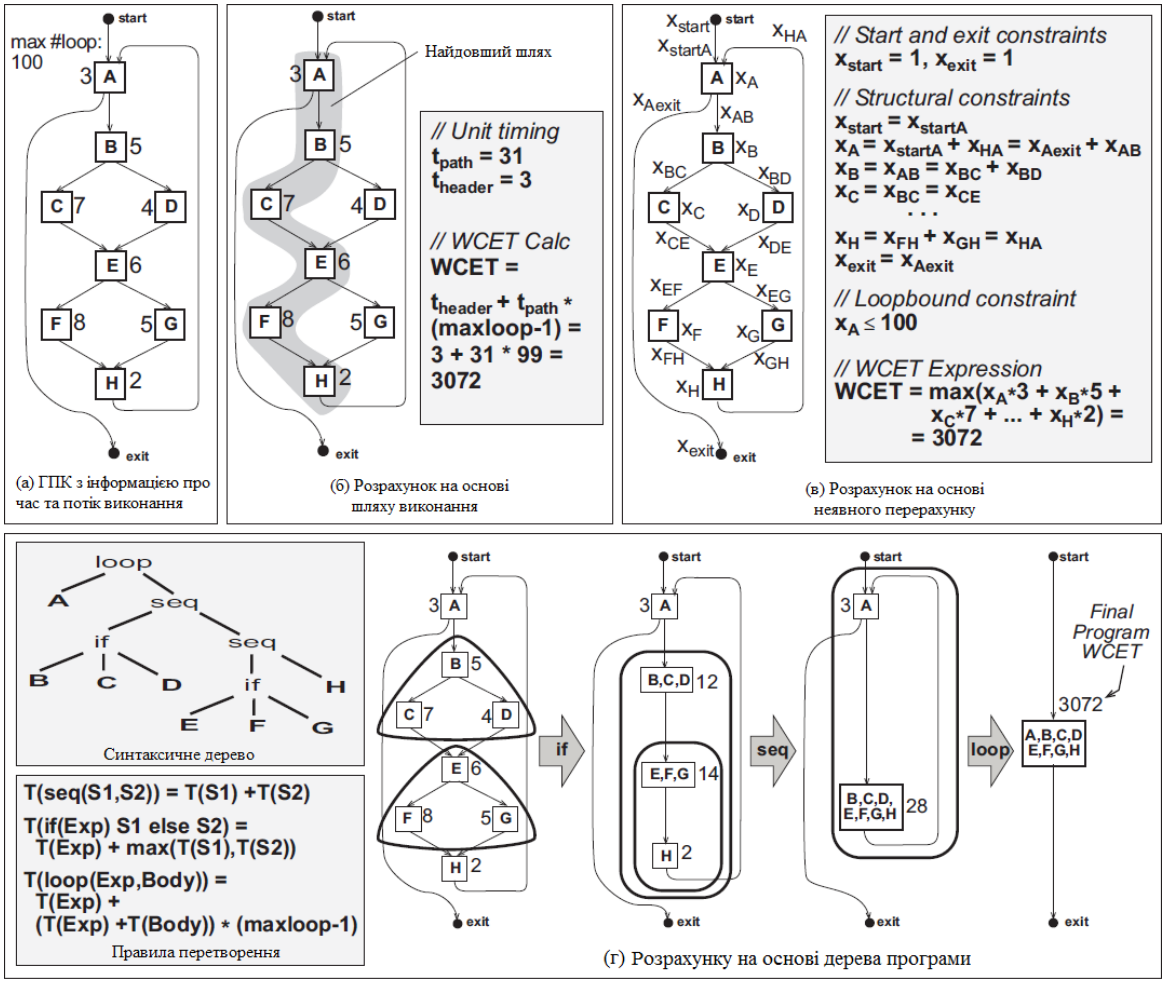


Рисунок 1.5 – Алгоритми розрахунку найбільшої тривалості виконання програмного коду: (а) – граф потоку керування з інформацією про час та потік виконання програми; (б) – розрахунок на підставі шляху виконання програми; (в) – розрахунок на підставі неявного перерахунку програми; (г) – розрахунок на підставі дерева програми

У роботах [7, 22, 61] розглянуто підходи, що використовують метод розрахунку на підставі дерева програми. У роботі [22] автором A. Colin досліджено ефективність застосування цього методу для визначення найбільшої тривалості виконання програми та прогнозування максимальної кількості циклів, що призведе до неї. У роботі [7] запропоновано підхід для оцінювання найбільшої тривалості виконання програми, що використовує часову абстракцію, яка містить детальну інформацію про тривалість виконання кожного шляху програми. У цій роботі експериментально доведено, що використання цього методу для вбудованих систем, які побудовані на

мікроконтролері з архітектурою RISC, дає змогу зменшити рівень переоцінювання значення WCET на 20%, що істотно підвищує рівень адекватності отриманих результатів. У роботі [61] наведено архітектуру програмного засобу, що виконує оцінювання найменшої та найбільшої тривалості виконання програмного коду, яка використовує метод розрахунку на підставі дерева програми. Недоліком запропонованого засобу є висока складність його адаптування для мікроконтролерів з архітектурою ARM та потребою удосконалення статичного аналізатора коду.

Метод розрахунку на підставі шляху виконання (*англ. path-based WCET calculation)* передбачає розрахунок тривалості виконання програмного коду усіма можливими шляхами виконання функції для однієї ітерації циклу, пошук серед можливих шляхів виконання програми шляху, що відповідає максимальній тривалості виконання та поєднання з інформацією про кількість циклів виконання фрагменту програми (рис. 1.5, б). Роботи [15, 74, 75] розглядають можливість застосування цього метода для аналізу тривалості виконання програмного коду мікроконтролерних вбудованих систем. У роботі [15] запропоновано підхід до розрахунку найменшої та найбільшої тривалості виконання програмного коду розробленого для аналізу великих фрагментів програми. У роботах [74, 75] автором F. Stappert запропоновано метод, що поєднує метод розрахунку на підставі шляху виконання програми, а також метод аналізу низькорівневих функцій. Експериментально доведено [61] ефективність цього методу завдяки незалежному аналізу функцій низького та високого рівня.

Метод неявного перерахунку шляху виконання програми, IPET (*англ. Implicit Path Enumeration Technique*) є одним із найчастіше використовуваних методів розрахунку. В ньому потік виконання програми та тривалість її виконання подають алгебраїчними формулами та/або логічними обмеженнями. Кожному базовому блоку і ребру графа присвоюють тривалість виконання (t_{entity}) та лічильник виконання (x_{entity}). Розрахунок найбільшої тривалості виконання виконується шляхом максимізації суми $\sum_{i \in entity} x_i \cdot t_i$ з урахуванням

обмежень, що відображають структуру програми і можливих потоків. Потім оцінювання найбільшої тривалості проводять з використанням цілочисельного лінійного програмування (ILP) або шляхом визначення обмежень. Внаслідок виконання таких дій отримують найбільшу кількість виконань кожного вузла чи ребра, а не явний шлях виконання програми, який призводить до найбільшої тривалості виконання.

Метод неявного перерахунку шляху виконання програми застосовують для оцінювання тривалості виконання програм вбудованих систем [35, 48, 53, 83]. У роботах [35, 83] досліджено ефективність застосування методу неявного перерахунку шляху виконання програми у поєднанні з низькорівневим аналізом конвеєра команд. Такий підхід до аналізу тривалості виконання програмного коду має високий ступінь адекватності результатів, проте його використання в практиці обмежене чималими витратами часу на аналіз програмного забезпечення.

У роботі [48] розглянуто можливість застосування методу неявного перерахунку шляху виконання програми для аналізу програм, які виконують цифрові сигнальні процесори, DSP (Digital Signal Processing). Досліджено можливість його адаптування для різних апаратних платформ.

Проведені дослідження продуктивності застосування методу неявного перерахунку шляху виконання програми, довели його значну ефективність для вбудованих систем, що використовують процесор Intel i960 [53, 66]. Окрім цього, експериментально встановлено [65], що ефективність цього методу є значно вища при аналізі відносно невеликих фрагментів програм.

Отже, зазначені методи статичного аналізу тривалості виконання програмного коду мають, як свої переваги, так низку істотних недоліків, що обмежує їх застосування. Основним недоліком статичних методів є те, що усі ці методи передбачають розроблення моделей апаратного забезпечення, що є найбільш трудомістким етапом. Складність цього етапу постійно збільшується, що зумовлено розвитком апаратного забезпечення, зокрема розвитком архітектури мікроконтролерів, а також недостатньою кількістю даних про їхні

часові характеристики. Окрім цього, велика складність апаратного забезпечення та "ручний" процес побудови моделей тягне за собою високу ймовірність внесення в неї помилок, які надалі призведуть до зниження адекватності отриманих результатів. Саме тому, методи статичного аналізу тривалості виконання програмного коду дають завищені результати, що є вимушеним кроком, для здійснення безпечного оцінювання тривалості виконання програмного коду.

1.3.2 Динамічні методи аналізу тривалості виконання програмного коду

Динамічні методи аналізу тривалості виконання програми передбачають проведення відповідних вимірювань безпосередньо на вбудованій системі. Класичним методом динамічного аналізу тривалості виконання програмного коду є відповідні вимірювання при різних значеннях вхідних даних, що часто є єдиним доступним методом такого аналізу, саме тому його найчастіше використовують. Загальним недоліком цього методу є складність пошуку вхідних даних, які призводять до найбільш тривалого виконання програми. Незважаючи на цей недолік, проведення вимірювань має потенційну перевагу над статичними методами. Позаяк програма виконується безпосередньо на апаратному забезпеченні вбудованої системи, то це дає змогу досліднику уникнути потреби створення моделей апаратного забезпечення, які вкрай потрібні для методів статичного аналізу.

Наявні динамічні методи аналізу тривалості виконання програми, які часто використовують практики, поділяють на такі, що виконують вимірювання за допомогою: осцилографа та логічного аналізатора, апаратного трасування, інтегрування додаткового програмного забезпечення, симуляторів вбудованої системи.

Метод аналізу тривалості виконання програмного коду з допомогою осцилографа та логічного аналізатора є базовим методом вимірювання. Осцилограф чи логічний аналізатор застосовують для спостереження за зміною

логічного рівня сигналу на виводі процесора, а також при виконанні певного сегмента програми [67, 90]. Отримані результати використовують для прогнозування тривалості виконання програмного коду. Основним недоліком цього методу є можливість тестування тільки "видимої" поведінки програми. Натомість, до його переваг варто віднести те, що він не передбачає внесення змін у вбудовану систему, яку тестують.

Метод аналізу тривалості виконання програмного коду, що виконуються з допомогою апаратного трасування, через інтерфейс відлагодження. Полягає у автоматизованій зміні вхідних даних і контролі результатів виконання програми через інтерфейс відлагодження. Найбільш відомими серед них є: ARM Embedded Trace Macrocell (ETM) [104], Joint Test Action Group (JTAG) [49], а також інтерфейс для відлагодження Nexus Debug Interfaces (NDI) [105].

У роботі [69] наведено архітектуру засобу для автоматизованого тестування вбудованих систем, яка передбачає тестування безпосередньо на ній через середовище розроблення ПЗ Eclipse IDE [106]. Це середовище зв’язується із вбудованою системою через інтерфейс відлагодження. В роботі вказано, як недолік слабкий рівень адаптації цього засобу для тестування тривалості виконання програмного коду, а також у ньому не передбачено можливість тестування окремої гілки програми. Вважаємо, що використання цього методу для тестування тривалості виконання програмного коду дає потенційну перевагу над іншими методами завдяки повному контролю над вбудованою системою та процесом тестування, без внесення змін у її програмне чи апаратне забезпечення. Недоліком цього методу є зниження швидкості виконання програми за рахунок витрати часу на надсилання даних через інтерфейс відлагодження. Однак, цей недолік є істотним при нефункціональному тестуванні. Саме тому його застосування для тестування тривалості виконання програмного коду без його удосконалення є небезпечним.

Метод аналізу тривалості виконання програмного коду, що виконуються з допомогою інтегрування додаткового програмного забезпечення широко використовуються при функціональному тестуванні ПЗ вбудованих систем.

У роботі [51] розглянуто можливості застосування цього методу тестування для тривалості виконання програмного коду, запропоновано архітектуру відповідного програмного засобу, а також процедуру вимірювання цієї тривалості. Для цього потрібно занести у текст програми контрольні точки, при досягненні яких вбудована система запускає/зупиняє таймер для вимірювання тривалості виконання програмного коду та надсилає результати вимірювання до персонального комп'ютера. Недоліком цього методу є те, що додавання тестового фреймворка до розробленого ПЗ призведе до істотної його зміни загалом. Окрім цього, інтегрування додаткового ПЗ та використання ним апаратних ресурсів (таймера та інтерфейсів комунікації з персональним комп'ютером) є не завжди можливим через істотні апаратні обмеження, що притаманні вбудованим системам.

Метод аналізу тривалості виконання програмного коду, що виконуються з допомогою симуляторів вбудованої системи дають змогу замінити реальне обладнання, однак їх розроблення та перевірка є дуже складним і трудомістким завданням.

У роботі [34] наведено симулятор процесора, розроблений для аналізу контекстної залежності (аналізу хеш-пам'яті) та розрахунку тривалості виконання програмного коду на підставі виконаного аналізу. Отримані результати є близькими до раніше опублікованих результатів, одержаних з використанням статичних методів. Автори цієї роботи стверджують, що розроблена методологія є безпечною, тому можна її застосовувати для аналізу тривалості виконання вбудованих систем жорсткого РЧ (див. розд. 1.3.1).

Недоліком використання симуляторів є висока складність та чимала тривалість процесу розроблення моделей процесора та апаратного забезпечення вбудованої системи. Спрощення моделей для збільшення швидкості їх побудови призведе до втрати точності отриманих результатів та їх непридатності для подальшого аналізу.

1.3.3 Гібридні методи аналізу тривалості виконання програмного коду

Гібридні методи аналізу тривалості виконання програмного коду поєднують переваги статичних і динамічних методів аналізу та передбачають виконання декількох етапів, а саме:

1) створення моделі програми;
2) вимірювання тривалості виконання обраних фрагментів програми;
3) об’єднання отриманої інформації та розрахунок найбільшої тривалості виконання програмного коду.

Створення моделі програми виконується на підставі статичного аналізу програмного коду в ході виконання якого у програмний код вноситься додаткова анотація, що представляє собою точки, які розбивають програму на декілька фрагментів для проведення вимірювання ТВПК. Для кожного фрагменту програми проводять вимірювання тривалості виконання N разів, після чого з отриманих результатів визначають найбільшу тривалість виконання, цю тривалість передають у створену модель програми, де її використовують для розрахунку найбільшої тривалості виконання програми.

У роботах [13, 12, 88] наведено засоби, що використовують методи гібридного аналізу тривалості виконання програмного коду. Проведені дослідження [12] довели, їх актуальність на реальних прикладах. Перевагою методу є відсутня потреба створення та аналізу моделі апаратного забезпечення процесора, що дає змогу зменшити витрачений час на аналіз тривалості виконання програмного коду. До недоліків цього методу належить: неможливість гарантувати, що отримані результати є безпечними (виміряне значення є максимальним).

Отже, проведений аналіз наявних методів статичного та динамічного аналізу тривалості виконання програмного коду дає змогу зробити такі висновки:

- Розроблення комплексних моделей, які враховують, як архітектуру конвеєра команд так і контекстну залежність програми за допомогою

аналізу хеш-пам’яті та хешу інструкцій є дуже трудомістким заданням до якого спостерігається підвищений інтерес дослідників.

- Зростання складності апаратного забезпечення призводить до зростання складності його моделей, а відтак до збільшення витрат часу на їх створення та аналіз.
- Розроблення нових динамічних методів аналізу тривалості виконання програмного коду дасть змогу підвищити точність отриманих результатів, а також зменшити час тестування за рахунок відсутності потреби виконувати розроблення моделей апаратного забезпечення.

1.4 Наявні програмні засоби для аналізу найбільшої тривалості виконання програмного коду

Для оцінювання можливостей сучасних вбудованих систем потрібно щоб аналіз найбільшої тривалості виконання програмного коду проводився з допомогою відповідних програмних засобів, наявних на ринку ПЗ. Порівняння отриманих результатів виконувались для таких функціональних характеристик:

- точність отриманих результатів;
- безпечність отриманих результатів;
- математичний апарат;
- підтримка декількох видів процесорів і апаратного забезпечення вбудованої системи;
- зручність використання.

Нижче наведено огляд та коротка характеристика відібраних систем для аналізу найбільшої тривалості виконання програмного коду, які широко застосовують під час тестування ТВПК.

Програмний засіб Bound-T розроблено компанією Tidorum Ltd за підтримки Європейського аерокосмічного агентства [75]. Цей програмний засіб призначений для статичного аналізу тривалості виконання програмного коду та визначення найбільшої тривалості виконання програмного коду. За даними розробника, засіб Bound-T підтримує декілька процесорів, а саме: Intel 8051 [107], ADSP-21020 [108], Atmel AVR [109] і ERC-32 SPARC [110].

Математичний апарат ґрунтується на підставі методу розрахунку за допомогою дерева програми без застосування аналізу стану конвеєра команд та хеш-пам’яті, саме тому реалізований метод завищує отримані результати, що робить оцінювання тривалості виконання програмного коду безпечним.

У програмному засобі Bound-T відсутній розвинутий графічний інтерфейс, що дещо ускладнює його використання, адже управління інструментом аналізу доступне тільки з командного рядка. Однак, за рахунок кросплатформеності він, може, працювати під управлінням ОС: Windows, Linux та OS/X.

Його недоліком є те, що компанія Tidorum Ltd зупинила розроблення та підтримку цього програмного засобу, що дуже погано для його кінцевих споживачів.

Програмний засіб aiT, розроблено компанією AbsInt, призначений для статичного аналізу тривалості виконання програмного коду та розрахунку найбільшої тривалості виконання програмного коду [85]. Цей програмний засіб побудований на підставі методу абстрактної інтерпретації, і з застосуванням аналізу стану конвеєра команд та хеш-пам’яті, що забезпечує високий рівень точності отриманих результатів та мінімальний рівень їх завищення. Саме тому його використовують такі компанії, як: Airbus, Daimler, NASA та інші.

Графічний інтерфейс користувача (рис. 1.6) є простим та інтуїтивно зрозумілим, характеризується хорошим рівнем візуалізації результатів. Прив’язка отриманих результатів до програмного коду дає змогу зменшити час на їх опрацювання.

До недоліків цього програмного засобу належить: завищення отриманих результатів, що притаманне усім засобам статичного аналізу програмного коду, а також висока вартість ліцензії на його використання.

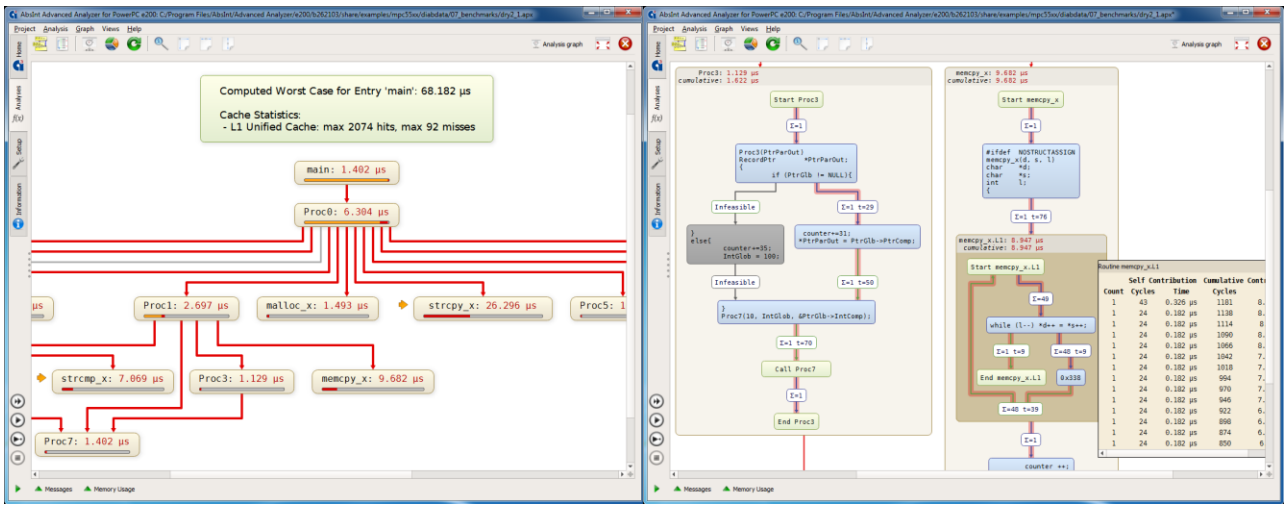


Рисунок 1.6 – Графічний інтерфейс програмного засобу aiT [76]

Програмний засіб RapiTime розроблено компанією RAPITA System Ltd, призначений для автоматизованого вимірювання тривалості виконання програмного коду вбудованих систем, які розробляють для автомобільної та авіа- галузі [72].

Цей програмний засіб побудовано з використанням методу динамічного аналізу з трасування, що забезпечує високу точність отриманих результатів, а також дає змогу виконання тестування безпосередньо на вбудованій системі. Програмний засіб підтримує широкий асортимент процесорів, для яких можна його застосовувати та інтегрується у більшість популярних середовищ розроблення програмного коду. Його використовують такі компанії, як: Airbus, Eesa, Embraer, Infineon, EADS Casa та інші.

Графічний інтерфейс системи (рис 1.7) функціонально схожий до своїх аналогів і дає змогу швидко провести дослідження тривалості виконання програмного коду. Їх візуальне подання дає змогу швидко ознайомитися з результатами дослідження.

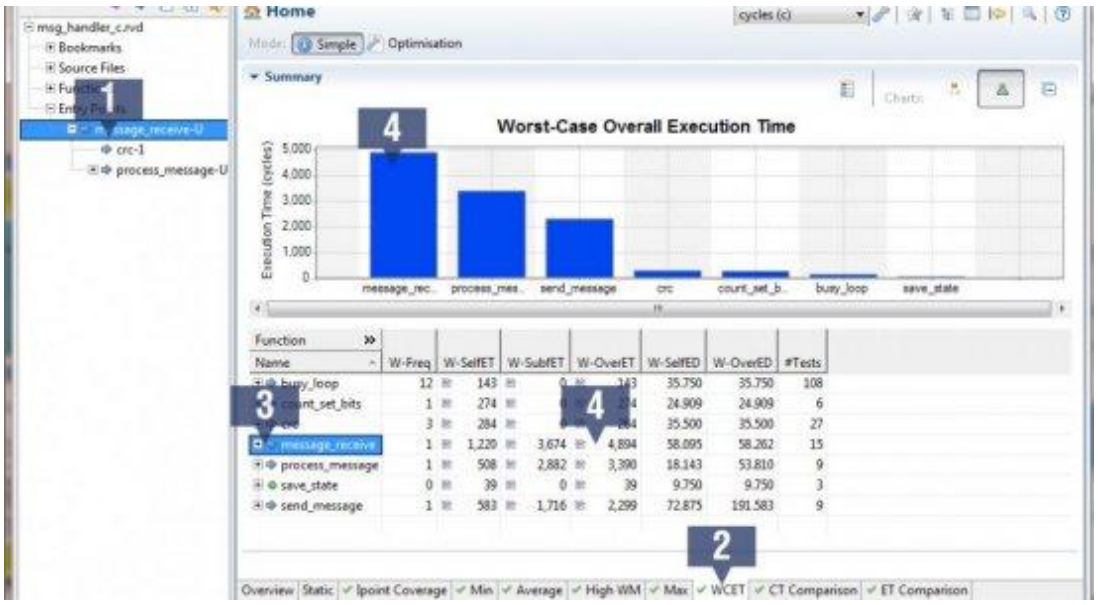


Рисунок 1.7 – Графічний інтерфейс програмного засобу RapiTime [72]

До недоліків цього програмного засобу належить: висока вартість ліцензії на його використання, а також несумісність з середовищем розроблення ПЗ вбудованих систем Keil ARM.

1.5 Висновки до розділу

1. Для проведення аналізу тривалості виконання програмного коду на практиці найчастіше використовують статичні методи аналізу, однак їх використання обмежується через зростанню складності процесу розроблення моделей апаратного забезпечення вбудованих систем, а відтак і зростанням ймовірності внесення помилки у розроблену модель. Розвиток апаратного забезпечення також призводить до зростання складності аналізу розроблених моделей. Саме будь-яке спрощення цих моделей призводить до зниження точності отриманих результатів.

2. Виявлено, що підвищення точності отриманих результатів без зростання складності моделі апаратного забезпечення можливе за рахунок проведення вимірювання тривалості виконання програмного коду низькорівневих функцій. Проте, цей підхід потребує подальшого вдосконалення наявних методів динамічного аналізу, що дасть змогу аналізувати усі гілки виконання програми без внесення змін у програмне чи апаратне забезпечення вбудованої системи.

3. Виявлено, що розроблені засоби аналізу тривалості виконання програмного коду використовують математичні моделі, які враховують стан конвеєра команд, чи хеш-пам’яті, однак не враховують вплив зовнішніх і внутрішніх чинників, таких як: температура навколишнього середовища, напруга живлення, старіння компонент.

4. Встановлено, що наявні засоби автоматизованого тестування ТВПК здебільшого використовують статичні методи аналізу, або їх розробляли для конкретного апаратного забезпечення. Тому актуальним є завдання створення засобу автоматизованого тестування тривалості виконання програмного коду, що не прив’язаний до конкретного апаратного забезпечення вбудованої системи, і забезпечує високий рівень адекватності отриманих результатів.

4. На підставі проведеного дослідження можна сформулювати такі напрямки дослідження:

а) підвищення рівня автоматизації процесу тестування тривалості виконання програмного коду;

б) удосконалення методів вимірювання тривалості виконання програмного коду шляхом підвищення точності отриманих результатів, а також усунення жорсткої прив’язки до апаратного забезпечення вбудованої системи.

в) розроблення моделей функціонування ПЗ, які дають змогу врахувати вплив зовнішніх та внутрішніх чинників на тривалість виконання програми, і в такий спосіб дає змогу формувати вимоги до середовища функціонування вбудованої системи, формувати план тестування та визначати періоду технічного обслуговування.

г) розроблення алгоритмічного і програмного забезпечення для автоматизованого тестування тривалості виконання програмного коду вбудованих систем жорсткого РЧ.

6. Вирішення поставлених завдань дасть змогу підвищити ефективність процесу тестування, шляхом підвищення точності отриманих результатів та зменшення часових витрат на його виконання.

РОЗДІЛ 2. МЕТОДИ ДИНАМІЧНОГО АНАЛІЗУ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ

У другому розділі сформульвано принципи побудови ПЗ для динамічного аналізу тривалості виконання програмного коду, що враховує специфіку сучасних мікроконтролерних вбудованих систем, а також результати дослідження наявних систем автоматизованого тестування тривалості виконання програмного коду, наведених в розд. 1.

Розглянуто методи динамічного аналізу відповідно визначеним принципів їх побудови. Наведено результати дослідження залежності похибки вимірювання ТВПК від тривалості виконання функції, а також відстані між ПК, який проводить тестування та вбудованою системою.

2.1 Принципи побудови програмного забезпечення для тестування тривалості виконання програмного коду

Першим кроком під час розробленні програмного засобу для автоматизованого тестування ТВПК вбудованих систем є формування концепції майбутньої системи та визначення основних принципів організації програмних модулів, що дає змогу впровадити цей засіб у виробничий процес. Під час формування цих принципів, окрім, результатів досліджень, проведених у розд.1, був використаний попередній досвід, набутий автором під час розроблення ПЗ вбудованих систем жорсткого РЧ.

Розроблено основні принципи, які визначають вимоги, архітектуру та особливості побудови та роботи програмного засобу для тестування ТВПК вбудованих систем, а саме:

1. *Тестування тривалості виконання програмного коду у режимі відлагодження чи трасування.* Програмний засіб повинен забезпечувати повний контроль над вбудованою системою упродовж виконання програми без інтегрування у її програмний код додаткових функцій для тестування. Окрім цього, такий підхід не передбачає використання вузько спеціалізованого

апаратного забезпечення, а тільки стандартний програматор, що використовує інтерфейс SWO [2] чи JTAG [49].

2. *Орієнтація на різне апаратне забезпечення, що базується на мікроконтролерах з архітектурою ARM.* Система має бути придатною для тестування різних вбудованих систем жорсткого РЧ, що працює під управлінням операційної системи РЧ. Підтримка мікроконтролерів з архітектурою ARM істотно підвищує гнучкість системи та охоплює тисячі моделей мікроконтролерів, що за даними [89] використовують 37% вбудованих систем, які випускають на сьогодні. Охоплення архітектури ARM забезпечить актуальність розробленої системи на найближчі 5 років, а відтак забезпечить постійне зростання кількості її потенційних користувачів.

3. *Відкритість до інтеграції різних алгоритмів вимірювання тривалості виконання програмного коду.* Важливою вимогою до програмного засобу є відкритість його архітектури для можливості легкої заміни алгоритму вимірювання тривалості виконання програмного коду. Водночас основною вимогою до алгоритму тестування є його самодостатність, тобто для його застосування не потрібно вносити зміни у програмне чи апаратне забезпечення вбудованої системи.

4. *Використання мережевих технологій.* Застосування мережевих протоколів та технологій дасть змогу зробити процес тестування віддаленим, що є корисним за браком доступу до усієї вбудованої системи, або за потреби, виконати тестування вбудованої системи, що працює у шкідливих умовах для життя людини, наприклад, підвищений рівень радіаційного випромінювання.

2.2 Метод динамічного аналізу тривалості виконання програмного коду мікроконтролерних вбудованих систем

В основу методу динамічного аналізу ТВПК покладено метод аналізу тривалості виконання програмного коду, що виконуються через інтерфейс відлагодження. Запропонований метод [69] передбачає виконання тестування ВС під час відлагодження програми у середовищі її розроблення. Перевагою такого методу є те, що для його застосування не потрібно використовувати

спеціалізоване апаратне забезпечення, або інтегрувати у ПЗ вбудованої системи додатковий код, який буде виконувати її тестування. Однак недоліком цього методу є те, що він впливає на ТВПК, а відтак вносить похибку у результати аналізу. Окрім цього, розроблений метод не забезпечує повний контроль над перебігом виконання програми, а відтак він не може забезпечити тестування окремої гілки програмної функції.

Розвиток методу динамічного аналізу полягає додаванні алгоритму тестування ТВПК, а також його адаптації для застосування з середовищем розроблення ПЗ вбудованих систем Keil µVision. В основі методу тестування ТВПК є вимірювання тривалості перебування між двома точками зупинки (breakpoint), що встановлені на початку та наприкінці фрагменту програми, тривалість якої потрібно виміряти. Точкою зупинки називається місце в програмі, яке з допомогою певного засобу дає змогу отримати інформацію про хід виконання програми [17].

Вимірювання ТВПК проводимо персональним комп’ютером, що виконує процедуру тестування за певним алгоритмом, який дає змогу забезпечити високу точність отриманих результатів без використання апаратних ресурсів вбудованої системи. Зв'язок вбудованої системи з персональним комп’ютером виконується через інтерфейс відлагодження, що дає змогу повністю контролювати процес виконання програми вбудованої системи. Застосування запропонованого методу для вбудованих систем, що побудовані на мікроконтролерах з архітектурою ARM, не змінює перебіг виконання програми за рахунок незалежного від ядра мікроконтролера модуля відлагодження ARM CoreSight Debug [8].

Для зменшення витрат часу на реалізацію та тестування алгоритмів обміну даними між персональним комп’ютером та вбудованою системою через інтерфейс відлагодження використовуємо програмний інтерфейс µVision Socket Interface API, що дає змогу керувати і відстежувати стан середовища розроблення ПЗ Keil µVision IDE з сторонніх програмних засобів [86]. Отже, в

такий спосіб отримуємо програмну реалізацію алгоритмів обміну даними у режимі відлагодження, яка сумісна з тисячами моделей мікроконтролерів.

Основними кроками методу тестування тривалості виконання програм є:

Крок 1. Пошук початкової адреси програмної функції, за якою вона розміщується у пам’яті програми вбудованої системи. На цьому кроці проводимо синтаксичний аналіз map-файлу, що генерується в процесі компіляції проекту. Цей файл містить імена усіх функцій, які увійшли до збірки, а також імена глобальних змінних і адреси, за якими вони розміщені у оперативній пам’яті.

Крок 2. Пошук кінцевої адреси функції. Проведення синтаксичного аналізу listing-файлу, що генерується в процесі компіляції ПЗ вбудованих систем. Listing-файл містить програмний код мовою С та його інтерпретацію асемблерним кодом. Шляхом статичного аналізу listing-файлу визначаємо кінцеву адресу функції.

Крок 3. Запуск режиму відлагодження проекту у середовищі Keil µVision, та початок виконання програми.

Крок 4. Встановлення точок зупинки за адресами, що відповідають початку та завершенню функції.

Крок 5. Встановлення значень глобальних змінних, що забезпечують перехід до виконання функції, яка підлягає тестуванню.

Крок 6. Запуск таймера у обробнику переривання, виклик якого зумовлений зупинкою виконання програми мікроконтролера у встановленій точці.

Крок 7. Надсилання команди продовження виконання програми.

Крок 8. Під час повторного виклику обробника переривання зупиняємо вимірювання тривалості виконання програмної функції та відображаємо отримані результати вимірювання.

Блок-схема алгоритму наведена на рис. 2.1.

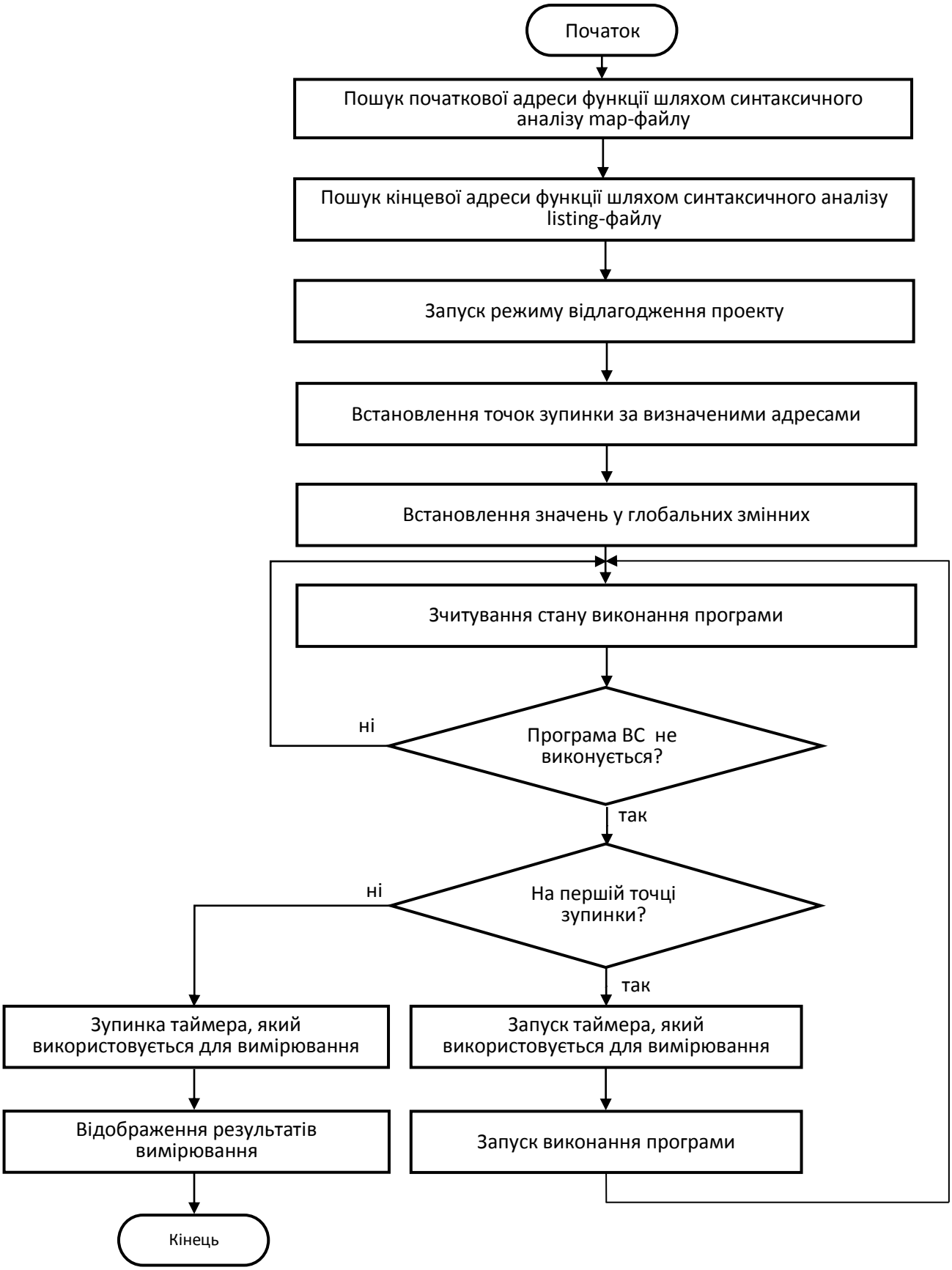


Рисунок 2.1 – Схема алгоритму реалізації метод динамічного тестування ТВПК мікроконтролерних вбудованих систем

Генерування різних значень локальних і глобальних змінних забезпечує можливість вимірювання тривалості виконання програми усіх гілок (забезпечує покриття повне покриття гілок програми), саме тому цей метод, може, використовуватися для вимірювання найбільшої тривалості виконання програмного коду. Однак, варто врахувати те що, запропонований метод динамічного тестування ТВПК не гарантує те, що виміряне значення є найбільшою тривалістю виконання програмного коду, а відтак збільшення кількості вимірювань призведе до підвищення ймовірності його виявлення, але не забезпечить стовідсоткового його виявлення, що є недоліком цього методу.

На підставі розробленого методу динамічного тестування ТВПК розроблено архітектуру програмного модуля (рис. 2.2).

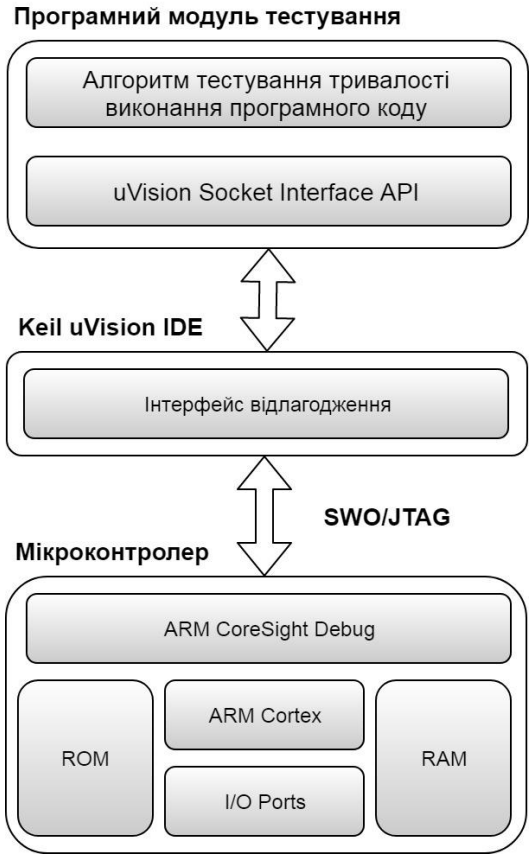


Рисунок 2.2 – Архітектура програмного модуля для тестування ТВПК

В основі архітектури програмного модуля лежать наведені вище принципи тестування ТВПК, що дало змогу зробити його у вигляді функціонально завершеної програмної компоненти, яка застосовуватись, як

окремо, так і в складі програмного засобу для автоматизованого тестування ТВПК.

2.3 Метод віддаленого тестування тривалості виконання програмного коду вбудованих систем

У багатьох випадках розробник ПЗ не має доступу до усієї вбудованої системи через її великі розміри, високу вартість чи масу, або через специфіку її застосування (підвищений рівень радіації, в умовах дії високого тиску, або високого рівня вібрацій). Тому розробники ПЗ працюють не з усією вбудованою системою, а тільки з її окремими частинами, або використовують симулятори її роботи. Подальше тестування вбудованої системи повинно виконуватися віддалено, що і призводить до потреби розроблення методу віддаленого тестування тривалості виконання програмного коду.

Розроблений методу враховує четвертий принцип тестування тривалості виконання програмного коду і передбачає виконання таких кроків: розроблення архітектури програмного модуля для віддаленого тестування, а також розроблення алгоритму його роботи.

2.3.1 Архітектура програмного модуля для віддаленого тестування тривалості виконання програмного коду вбудованих систем

Під час розроблення архітектури програмного модуля для віддаленого тестування ТВПК потрібно врахувати те, що швидкість Інтернет-з’єднання в реальних умовах експлуатації вбудованої системи, як правило, є низькою, а відтак метод тестування повинен характеризуватися максимальною простотою та мінімальним навантаженням на канал зв’язку.

Метод віддаленого тестування передбачає використання двох персональних комп’ютерів, що використовують клієнт-серверну архітектуру зв’язку [19]. Клієнт (персональний комп'ютер, ПК1) – виконує процедуру тестування, надсилає тестові дані та інструкції до сервера, а також виконує вимірювання тривалості виконання програми. Сервер (персональний комп'ютер, ПК2) – здійснює контроль за виконанням програми на вбудованій

системі, яка під'єднана до нього через інтерфейс відлагодження. Сервер (TCP сервер) створюється середовищем розроблення Keil µVision, що дає змогу інженерам – тестувальникам використовувати стандартні можливості програмного модуля відлагодження: керування процесом виконання коду, контроль та моніторинг за значеннями змінних, встановлення точок зупинок тощо.

Блок-схема процесу віддаленого тестування ТВПК зображена на рис. 2.3.

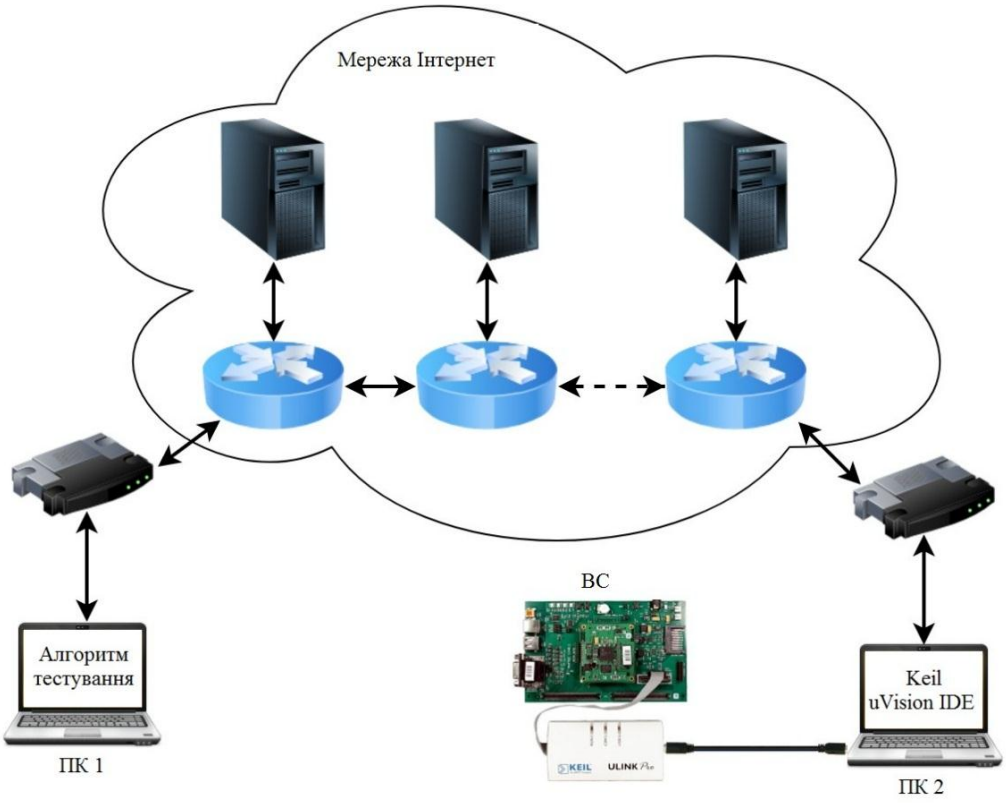


Рисунок 2.3 – Блок-схема процесу віддаленого ТВПК

2.3.2 Алгоритм віддаленого тестування тривалості виконання програмного коду

Тестування вбудованої системи через мережу Інтернет призводить до додавання похибки вимірювання, яка зумовлена затримкою надсилання пакетів даних через мережу Інтернет. Саме тому, до методу динамічного тестування додається ще один крок, який виконує процедуру компенсації похибки вимірювання. Для цього проводиться багатократне вимірювання затримки надсилання пакетів з використанням протоколу ICMP [64], розрахунок

середнього арифметичного значення затримки та компенсування отриманих результатів. Схема алгоритму віддаленого тестування тривалості виконання програмного коду наведена на рис. 2.4.

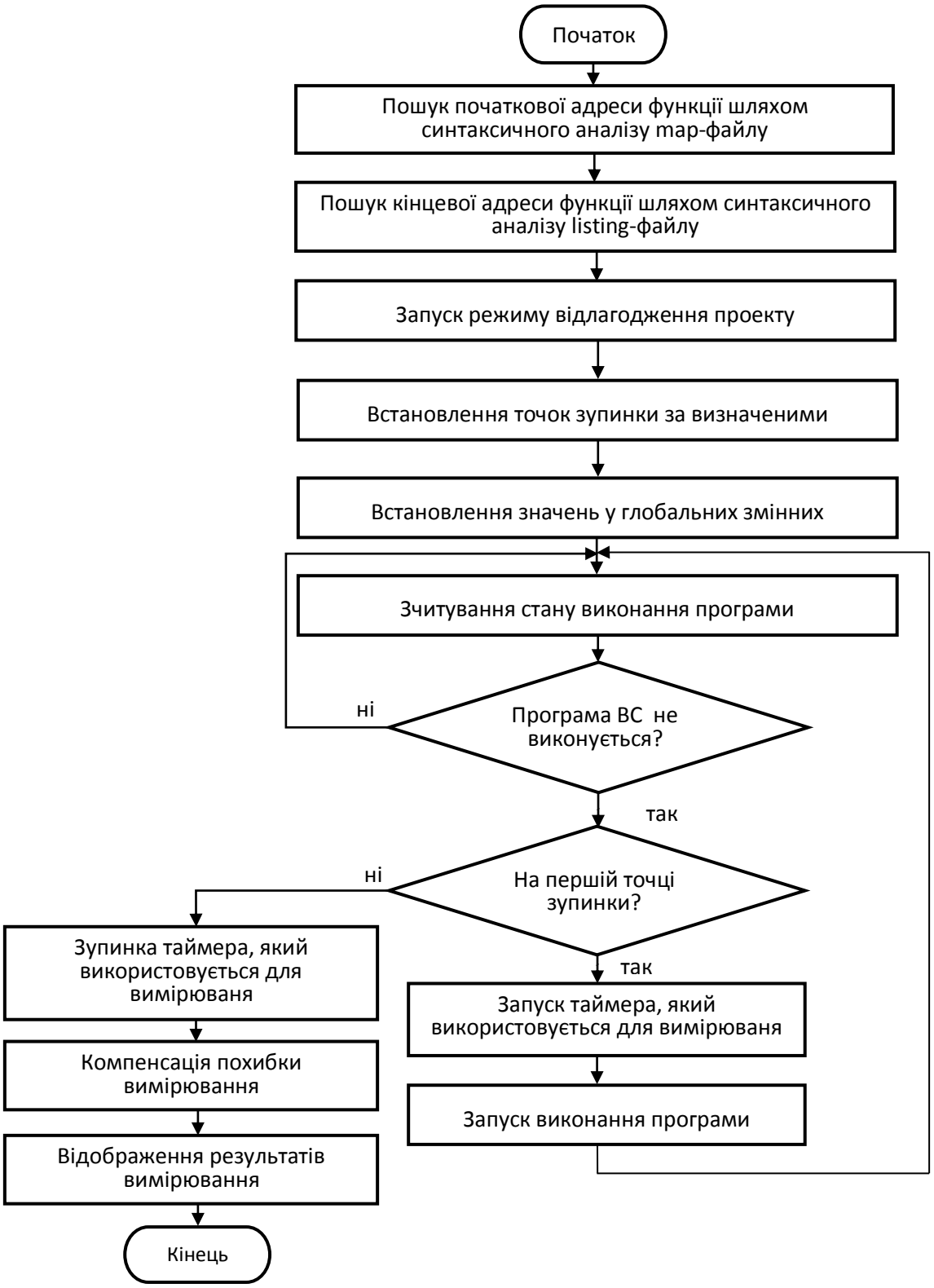


Рисунок 2.4 – Схема алгоритму реалізації методу віддаленого тестування ТВПК мікроконтролерних вбудованих

2.4 Верифікація методів тестування тривалості виконання програмного коду

Під верифікацією методів тестування будемо розуміти спосіб підтвердження за допомогою доказів певних теоретичних положень шляхом їх зіставлення з емпіричними даними. Відповідно, для підтвердження коректності отриманих результатів з використанням запропонованих методів тестування тривалості виконання програмного коду можна використати емпіричні дані про ТВПК, отримані з використанням інших методів оцінювання. Для максимально коректної верифікації методів тестування тривалості виконання програмного коду варто використати функцію з відомою тривалістю виконання. Тривалість виконання опорної функції має змінюватися в широких межах, що забезпечить достатню статистичну вибірку емпіричних даних, що отримані під час експерименту. Як опорну функцію обрано функцію, яка реалізує почергову зміну рівня на виводі мікроконтролера, з певним інтервалом.

Текст програми має такий вигляд:

```
void ChangeLevel(void)
{
	GPIOD->ODR = 0x9000;
	delay(1000);
	GPIOD->ODR = 0x0000;
	delay(1000);
}

void SysTick_Handler(void)
{
	 ticks_delay++;
}

void delay(uint32_t milliseconds)
{
	uint32_t start = ticks_delay;
	while((ticks_delay - start) < milliseconds);
}
```

Функція ChangeLevel використовує функції затримки від апаратного таймера, а відтак їх тривалість є завжди постійною та не залежить від кількості

програмних інструкцій, тривалості зчитування інструкції з пам’яті програм, стану конвеєра команд, чи стану оперативної пам’яті. Окрім цього, простота цієї функції дає змогу використати метод вимірювання тривалості виконання програмного коду за допомогою осцилографа, як джерело опорної інформації.

Проведені нами дослідження були проведені так, щоб імітувати роботу функцій з різною ТВПК, саме тому функція "delay" викликалася з різними вхідними значеннями, що змінювалися в діапазоні 1 мс – 10 с. Результати тестування тривалості виконання програмного коду з використанням розробленого методу, наведено в табл. 2.1

Таблиця 2.1 – Виміряні показники тривалості виконання програми отримані за допомогою методу динамічного тестування ТВПК

Інтервал зміни логічного рівня, с	Виміряна тривалість з допомогою осцилографа, с	Тривалість виконання функції при використанні запропонованого методу, с	Відносна похибка вимірювання, %
0,001	0,001	0,0012	16,667
0,002	0,002	0,0023	13,043
0,005	0,005	0,0053	6,367
0,01	0,01	0,0102	1,961
0,02	0,02	0,0202	1,137
0,05	0,05	0,0504	0,853
0,1	0,1	0,1005	0,498
0,2	0,2	0,2002	0,100
0,5	0,5	0,5003	0,060
1,0	1,0	1,0002	0,020
2,0	2,0	2,0003	0,015
5,0	5,0	5,0003	0,007
10,0	10,0	10,0005	0,005

Дані табл. 2.1 показують, що запропонований метод динамічного тестування характеризується похибкою вимірювання, яка перебувала в межах 0,2 – 0,5 мс і змінюються випадково, а відтак ця похибка вважається похибкою методу вимірювання, яку неможливо скомпенсувати. Графік залежності відносної похибки вимірювання від ТВПК наведено на рис. 2.5.

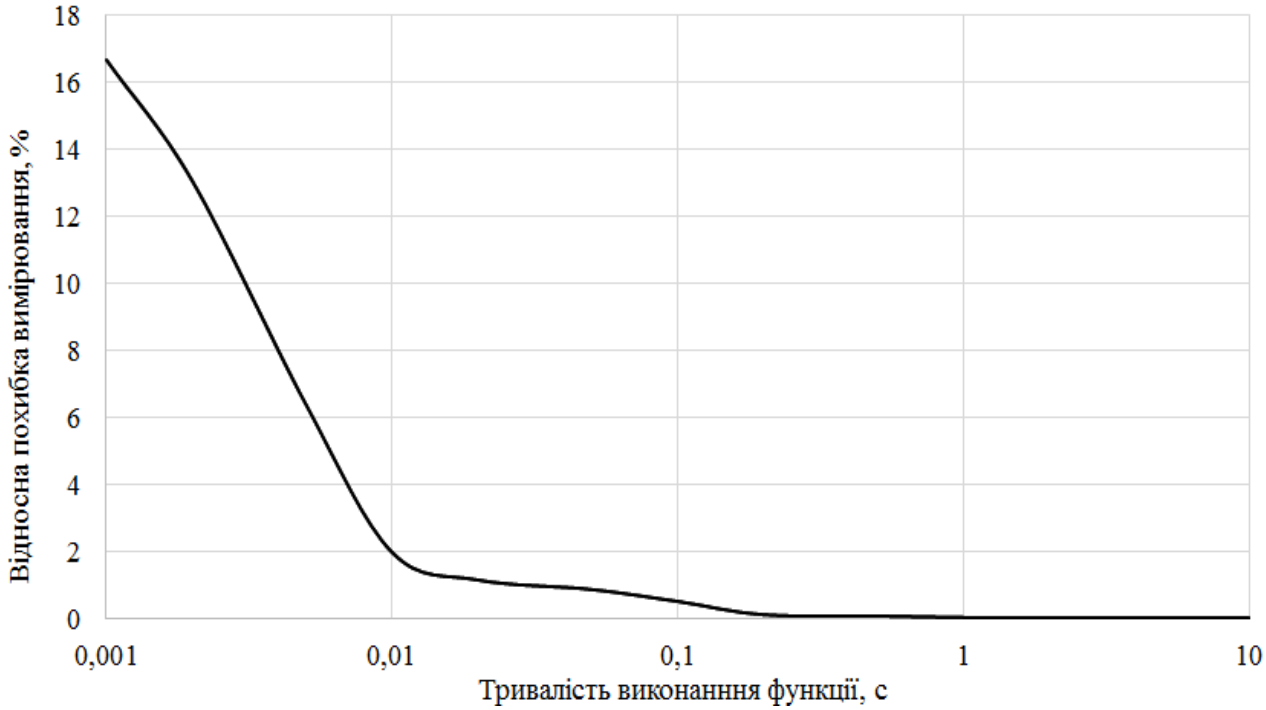


Рисунок 2.5 – Залежність відносної похибки вимірювання тривалості виконання програмного коду від ТПВК

Як бачимо з рис. 2.5 відносна похибка вимірювання ТПВК збільшується під час зменшення тривалості виконання функції, однак ця похибка вимірювання завищує отримані результати, а відтак робить їх безпечними. Саме тому цей метод, може, використовуватись при дослідженні будь-якої тривалості виконання програми за умови, що отримана точність задовольняє інженерів з якості ПЗ.

Наступним кроком є проведення аналогічного спостереження для методу віддаленого тестування ТВПК. Це дослідження проводили при різних відстанях між персональним комп’ютером, який виконує алгоритм тестування та вбудованою системою, що дає змогу оцінити залежність похибки вимірювання від відстані між вбудованою системою та ПК, що її тестує. Результати дослідження наведені у табл. 2.2-2.4. Залежність похибки вимірювання від відстані між вбудованою системою та ПК, який виконує алгоритм тестування, показана на рис. 2.6.

Таблиця 2.2 – Виміряні показники тривалості виконання програми отримані за допомогою методу віддаленого тестування (при відстані 10 м)

Інтервал зміни логічного рівня, с	Виміряна тривалість з допомогою осцилографа, с	Тривалість виконання функції при використанні запропонованого методу, с	Відносна похибка вимірювання, %
0,001	0,001	0,038	66,667
0,002	0,002	0,026	54,545
0,005	0,005	0,04	28,571
0,01	0,01	0,053	18,699
0,02	0,02	0,051	9,091
0,05	0,05	0,074	3,846
0,1	0,1	0,107	2,057
0,2	0,2	0,222	0,962
0,5	0,5	0,532	0,299
1,0	1,0	1,047	0,200
2,0	2,0	2,074	0,125
5,0	5,0	5,036	0,042
10,0	10,0	10,057	0,021

Таблиця 2.3 – Виміряні показники тривалості виконання програми отримані за допомогою методу віддаленого тестування (при відстані 1 км)

Інтервал зміни логічного рівня, с	Виміряна тривалість з допомогою осцилографа, с	Тривалість виконання функції при використанні запропонованого методу, с	Відносна похибка вимірювання, %
0,001	0,001	0,038	65,517
0,002	0,002	0,026	54,023
0,005	0,005	0,058	30,556
0,01	0,01	0,033	19,355
0,02	0,02	0,054	11,894
0,05	0,05	0,099	4,398
0,1	0,1	0,157	2,468
0,2	0,2	0,223	1,119
0,5	0,5	0,535	0,321
1,0	1,0	1,047	0,212
2,0	2,0	2,079	0,132
5,0	5,0	5,035	0,047
10,0	10,0	10,056	0,023

Таблиця 2.4 – Виміряні показники тривалості виконання програми отримані за допомогою методу віддаленого тестування (при відстані 1500 км)

Інтервал зміни логічного рівня, с	Виміряна тривалість з допомого осцилографа, с	Тривалість виконання функції при використанні запропонованого методу, с	Відносна похибка вимірювання, %
0,001	0,001	0,038	67,846
0,002	0,002	0,027	60,159
0,005	0,005	0,040	29,972
0,01	0,01	0,054	20,255
0,02	0,02	0,051	9,910
0,05	0,05	0,057	4,031
0,1	0,1	0,155	2,724
0,2	0,2	0,223	1,109
0,5	0,5	0,533	0,385
1,0	1,0	1,047	0,231
2,0	2,0	2,075	0,132
5,0	5,0	5,036	0,047
10,0	10,0	10,057	0,023

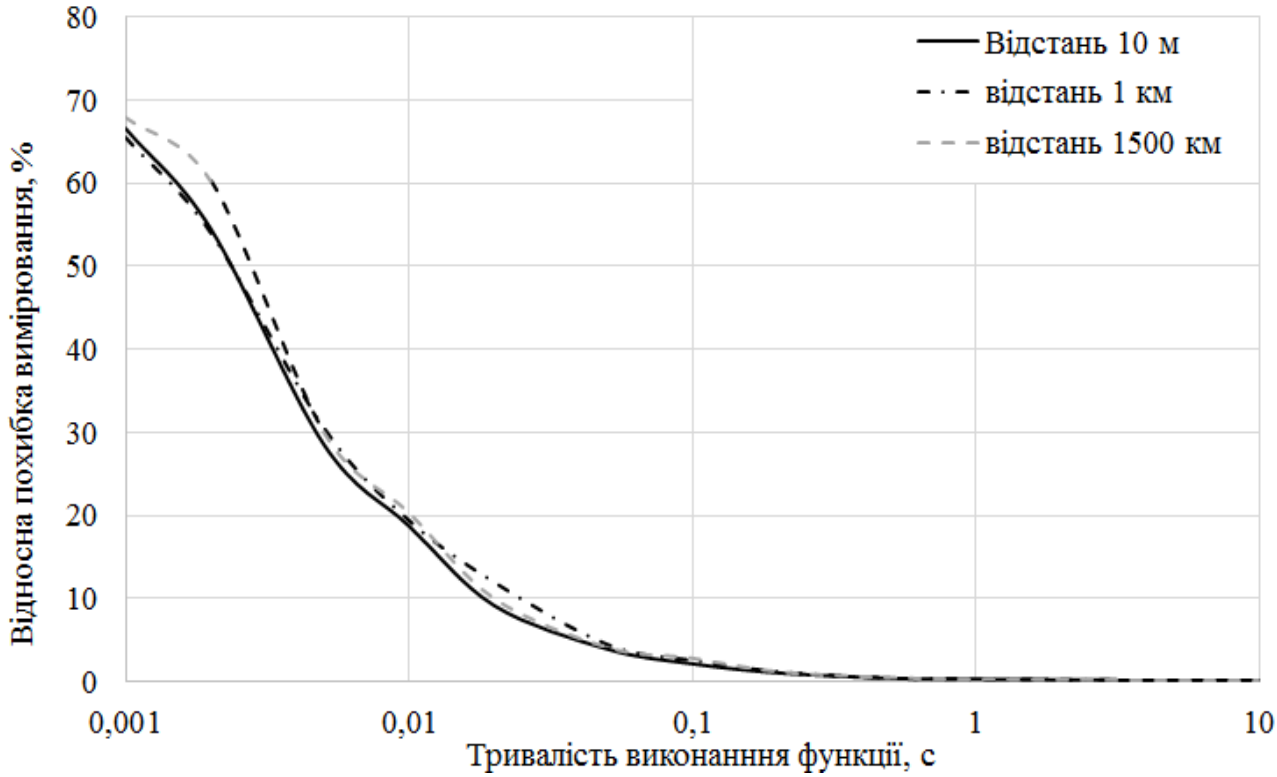


Рисунок 2.6 – Залежність відносної похибки вимірювання тривалості виконання програмного коду від тривалості виконання функції

Дані табл. 2.2 – 2.4 показують, що запропонований метод віддаленого тестування тривалості виконання програмного коду має більшу похибку вимірювання порівнюючи з методом динамічного тестування завдяки затримки надсилання пакетів даних через мережу Інтернет. Ця похибка є випадковою, а відтак її неможливо повністю компенсувати при використанні середньої затримки надсилання пакетів.

Як і метод динамічного тестування, так і метод віддаленого тестування завищує отримані результати, а відтак його можна використовувати при вимірюванні тривалості виконання будь-яких функцій. Однак, згідно з результатами, які подані на рис. 2.6, цей метод доцільно застосовувати при вимірюванні тривалості виконання функцій, тривалість яких більша за 100 мс.

2.5 Висновки до розділу

1) Визначено та обґрунтовано принципи побудови програмного засобу для тестування ТВПК, реалізація яких дасть змогу розробити належне ПЗ, що характеризується високим рівнем інтеграції з іншими програмними засобами та можливістю використання у вбудованих системах, що побудовані на мікроконтролера з архітектурою ARM.

2) Отримав подальший розвиток метод динамічного тестування ТВПК, який використовує інтерфейс відлагодження та базується на вимірюванні тривалості виконання коду між двома точками зупинки.

3) Встановлено істотне скорочення часових витрат на аналіз ТВПК порівнюючи з відповідними статичними методами аналізу ТВПК, що зумовлено відсутністю потреби розроблення складних моделей апаратного забезпечення.

4) Під час виконання дослідження розроблених методів виявлено, що:

а) їх можна застосовувати для аналізу ТВПК мікроконтролерних вбудованих систем жорсткого РЧ;

б) вони характеризуються високим рівнем автоматизації, що дає змогу зменшити часові витрати на тестування вбудованої системи;

в) метод віддаленого тестування вносить додаткову похибку, яка зумовлена затримкою надсилання даних через мережу Інтернет. Зменшення її рівня можливе завдяки зменшення кількості чинників, що впливають на випадковий складник похибки, а саме – зменшення впливу затримки надсилання пакету даних через мережу Інтернет шляхом використання допоміжних алгоритмів, що забезпечують фіксований маршрут відправлення пакетів;

г) подальше підвищення ефективності застосування методів динамічного аналізу тривалості виконання програмного коду можливий завдяки використання плану тестування функцій, що побудований з урахуванням поведінки периферійних пристроїв.

Основні результати розділу 2 опубліковані в працях [19, 98].

РОЗДІЛ 3. ОЦІНЮВАННЯ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ З УРАХУВАННЯМ ЧАСОВОЇ ПОВЕДІНКИ ПЕРИФЕРІЙНИХ ПРИСТРОЇВ

У розділі розроблено модель функціонування ПЗ, де вперше використано інформацію про поведінку периферійних пристроїв, що входять до складу вбудованої системи. На підставі запропонованої моделі удосконалено метод та алгоритм формування плану тестування вбудованої системи. Оцінено обчислювальну складність розробленого методу генерування плану тестування за кількістю операцій. Здійснено оцінювання тривалості формування плану тестування для проектів з різною кількістю функцій.

3.1 Модель функціонування програмного забезпечення, що ґрунтується на поведінці периферійних пристроїв

Модель функціонування ПЗ – це модель програмного коду, що відображає процес його виконання на вбудованій системі та деякий набір характеристик. Зазвичай модель функціонування ПЗ наводять у вигляді зваженого графа, вершинами якого є компоненти програми, а зваженими ребрами – ймовірності передавання управління між ними [25]. Однак для оцінювання тривалості виконання програмного коду, як модель використовують граф потоку керування CFG (англ. Control flow graph), вершинами якого є програмні інструкції, ребра відображають потік управління між ними.

Наявні моделі функціонування ПЗ, що використовуються під час аналізу тривалості виконання програми, враховують тільки архітектуру мікроконтролера та його стан. На жаль, цей підхід має низький рівень адекватності, адже будь-яка сучасна вбудована система містить багато компонентів (периферійних пристроїв) час відгуку яких впливає на ТВПК. Величина внесеної затримки залежить від периферійного пристрою та типу операції, що він виконує. Тому важливим завданням є врахування поведінки

периферійних пристроїв, що дасть змогу підвищити рівень адекватності отриманих результатів.

Для вирішення описаних вище завдань в роботі використано математичну модель функціонування ПЗ у вигляді орієнтованого графу $G=\{V, P\}$, де V – множина програмних інструкцій, P – множина переходів між ними. Процес виконання ПЗ зображено за допомогою пройдених шляхів такого графу, кожну вершину якого подано, як програмну інструкцію, а ребра відповідають послідовностям їх виклику.

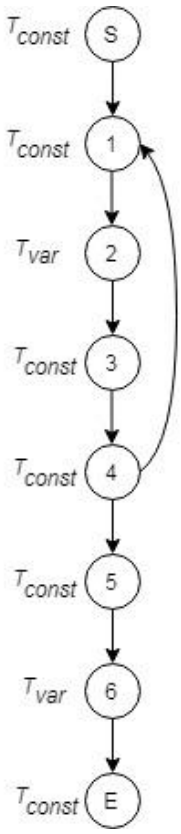
Основні позначення:

– $V_i^{periph\_changed}$ – множина програмних інструкцій, тривалість виконання яких залежить від поведінки зовнішніх периферійних пристроїв, може, змінюватися випадково в межах $T_{min} - T_{max}$.

– $V_i^{internal\_const}$– множина програмних інструкцій, тривалість виконання яких залежить від поведінки мікроконтролера, яка є постійною.

Тоді кожен вузол графу V_i є набором відповідних множин $(V_i^{periph\_changed}, V_i^{internal\_const})$ $i = 1, \overline{m}$ що характеризується і-ю тривалістю їх виконання, а також мінімальною, максимальною тривалістю виконання вузла і відповідним середньоквадратичним відхиленням. Кожен шлях виконання програми є сумою тривалостей виконання програмних інструкцій.

Приклад. Нехай вбудована система складається з таких компонент: мікроконтролер, аналого-цифровий перетворювач та флеш пам'ять. ПЗ вбудованої системи виконує вимірювання температури кожних 100 мс, усереднює виміряні результати та записує їх на флеш пам'ять. На рис. 3.1 наведено текст програми, а також модель функціонування ПЗ у вигляді графу потоку керування. Окрім цього, нанесено додаткове позначення, що характеризує залежність тривалості виконання програмної інструкції (тривалість перебування у вершині графу) від поведінки периферійних пристроїв.




```
S   void MeasureFromADC (void)
    {
        float average,SampleBuffer[10];
        short Chanel;
        byte  Counter;

        for (Counter = 0; Counter < 10; Counter++)
        {
1           Chanel = ADS_Read(ADSCON_CH1);
2           while(SPI_Get_Flag(SPI1, SPI_FLAG_BSY) == SET);
3           SampleBuffer[Counter] = ADC_code2temp(Chanel);
4           average += SampleBuffer[Counter];
        }
5       average = average/10;
6       WriteByteToTheMemory(Address, &average, 4);
E   }
```

Рисунок 3.1 – Приклад моделі функціонування програми для однієї функції вбудованої системи

Внаслідок урахування впливу зовнішніх периферійних пристроїв на ТВПК, розроблена модель більш точно описує поведінку ПЗ вбудованої системи.

3.2 Метод статичного аналізу тривалості виконання програмного коду для мікроконтролерних вбудованих систем

Одним із основних етапів формування плану тестування є статичний аналіз програмного коду, адже точність отриманих результатів безпосередньо впливає на ефективність його застосування. Для підвищення точності статичного аналізу тривалості виконання програмного коду розроблено метод, який на відміну від наявних методів враховує швидкість обміну даними між внутрішніми модулями мікроконтролера та швидкість надсилання даних через інтерфейси зв’язку.

Розроблений метод передбачає послідовне виконання декількох етапів аналізу, а саме: визначення налаштувань мікроконтролера на підставі

програмного коду, визначення взаємозв'язків між шинами синхронізації та внутрішніми модулями, визначення налаштувань інтерфейсів зв'язку з зовнішніми периферійними пристроями, розрахунок тривалості виконання програмного коду [144].

1. *Визначення налаштувань мікроконтролера на підставі програмного коду* потрібне тому, що за замовчуванням усі внутрішні модулі мікроконтролера вимкнені, а частота роботи його ядра є мінімальною. Відтак, процес увімкнення потрібних периферійних модулів, налаштування їхньої швидкості роботи, а також налаштування частоти роботи ядра мікроконтролера виконується при кожному запуску програми. Для визначення частоти, на якій буде працювати мікроконтролер, аналізуємо текст функції, що виконує його налаштування. Розрахунок частоти, з якою працює мікроконтролер, проводиться за формулою:

$$f_{CPU} = \frac{f_0}{PLL_M} \cdot \frac{PLL_N}{PLL_P}, \quad (3.1)$$

де: f_O – частота кварцового генератора; PLL_M, PLL_P – коефіцієнти ділення частоти кварцового генератора; PLL_N – коефіцієнт множення частоти кварцового генератора.

Розрахунок частоти синхросигналу на шинах APB проводиться за формулою:

$$f_{APB} = \frac{f_{CPU}}{APB_P}, \quad (3.2)$$

де APB_P – коефіцієнт ділення частоти.

2. *Визначення взаємозв'язків між шинами синхронізації та внутрішніми модулями* передбачає проведення аналізу даних документації для обраного мікроконтролера, зокрема розділу, що містить інформацію про можливі варіанти налаштування шин синхронізації та взаємозв'язок між шинами синхронізації та внутрішніми модулями. Аналіз цього розділу проводиться розробниками ПЗ вбудованих систем у ході його створення. Відтак, для визначення взаємозв'язків між шинами синхронізації та внутрішніми модулями достатньо провести синтаксичний аналіз тексту програми, що відповідає за

увімкнення тактування внутрішнього модуля. Текст програми має такий вигляд:

RCC_APB1PeriphClockCmd(*RCC_APB1Periph_PPPx*, *ENABLE*),

де: RCC_APB1PeriphClockCmd() – функція увімкнення тактування внутрішнього модуля, який під'єднаний до шини синхронізації APB; PPPx – назва периферійного модуля, ENABLE – параметр функції, що відповідає за ввімкнення тактування.

3. Визначення налаштувань інтерфейсів зв'язку із зовнішніми периферійними пристроями виконується за допомогою аналізу тексту програми, який виконує налаштування інтерфейсів зв'язку на відповідний режим роботи. Також, на цьому етапі виконується розрахунок тривалості надсилання одного байту даних через відповідний інтерфейс зв'язку.

Тривалість надсилання одного байту даних через інтерфейс SPI оцінюється за формулою:

$$t_{SPI} = t_{APB} + t_{bSPI} = 8 \cdot \frac{f_{APB} + f_{SPI}}{f_{APB} \cdot f_{SPI}}, \quad (3.3)$$

де: t_{APB} – тривалість надсилання одного байту до внутрішнього модуля SPI; t_{bSPI} – тривалість надсилання одного байту через інтерфейс SPI; f_{SPI} – робоча частота інтерфейсу SPI.

Тривалість надсилання одного байту даних через інтерфейс I2C проводиться за формулою:

$$t_{I2C} = t_{APB} + t_{bI2C} = \frac{8}{f_{APB}} + \frac{10}{f_{I2C}}, \quad (3.4)$$

де: t_{bI2C} – тривалість надсилання одного байту через інтерфейс I2C, що включає надсилання двох додаткових біт синхронізації; f_{I2C} – робоча частота інтерфейсу I2C.

Тривалість надсилання одного байту даних через інтерфейс UART проводиться за формулою:

$$t_{UART} = t_{APB} + t_{bUART} = \frac{M}{f_{APB}} + \frac{N}{f_{UART}}, \quad (3.5)$$

де: t_{bUART} – тривалість надсилання одного байту через інтерфейс UART, що включає надсилання N додаткових бітів; M – кількість інформаційних бітів у повідомленні, N – кількість додаткових бітів, що визначається конфігурацією інтерфейсу; f_{UART} – робоча частота інтерфейсу UART.

Типове повідомлення, що надсилається через інтерфейс UART зображене на рис. 3.2. Окрім інформаційних бітів, що надсилаються через інтерфейс також додаються мітки для синхронізації, які приймач автоматично видаляє з повідомлення. До цих міток належать один стартовий та 1 або 2 стоп бітів. Крім цього інтерфейс UART передбачає налаштування кількості інформаційних бітів, які передаються в повідомленні. Розмір цього повідомлення, може, змінюватися від 5 до 9 біт. Для підвищення завадостійкості повідомлення, що передається, може містити ще один додатковий біт перевірки парності.

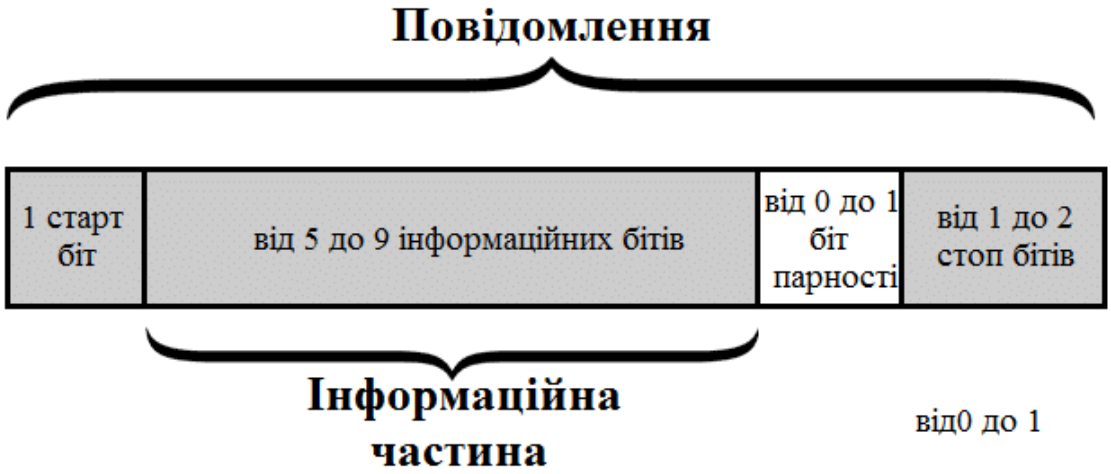


Рисунок 3.2 – Приклад повідомлення, що передається через інтерфейс UART

4. Розрахунок тривалості виконання програмного коду передбачає виконання побудови графу потоку керування, аналізу відповідності тексту програми і асемблерного коду, а також розрахунку ТВПК для кожної гілки.

4.1. Алгоритм побудови графу потоку керування за текстом програми.

Граф потоку керування є обов’язковим під час аналізу ТВПК адже він дає змогу визначити кількість гілок всередині кожної програмної функції, умови переходу у кожну гілку, а також аналізувати умови виходу з циклів програми.

Розроблений нами алгоритм [100] передбачає виконання чотирьох кроків: попередня обробка тексту програми, визначення кількості вершин графу, визначення кількості ребер у графі, заповнення матриці на основі визначених взаємозв'язків між вершинами графу. Процесу побудови графу потоку керування зображено на рис. 3.3.

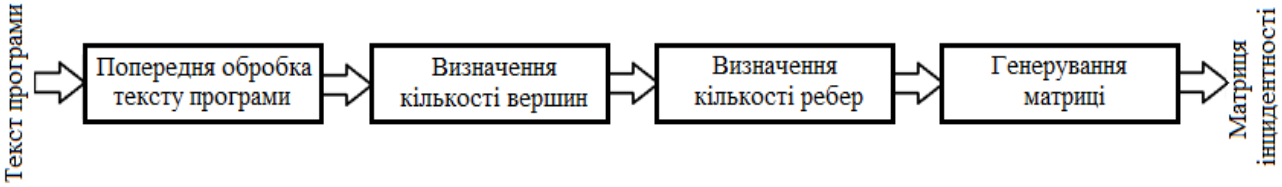


Рисунок 3.3 – Схема процесу побудови графу потоку керування за текстом програми, мовою С

Крок 1. Попередня обробка тексту програми. Під час виконання цього кроку з тексту програми видаляються коментарі, порожні рядки, а також проводиться модифікація тексту програми, якщо виклик функції або умову галуження записано у декілька рядків.

Крок 2. Визначення кількості вершин графу. Цей крок передбачає виконання послідовного нумерування усіх рядків тексту програми за винятком таких випадків:

- рядок тексту програми містить лише спеціальний символ "{" або "}";
- рядок тексту програми містить ідентифікатор: case, default, break, return, else, else if;
- рядок тексту програми містить ініціалізацію змінної чи екземпляра структури;

Крок 3. Визначення кількості ребер у графі. Виконання цього кроку передбачає нумерування ребер на основі синтаксичного аналізу тексту програми, а саме пошуку у ньому ідентифікаторів галуження (if, switch, else, else if), ідентифікаторів циклу (for, do, while) та ідентифікаторів слідування.

Визначення кількості ребер відбувається згідно з такими правилами:

- лічильник ребер збільшується на одиницю, якщо рядок тексту програми відповідає обчислювальному процесу – слідування;

- лічильник ребер збільшується на одиницю, якщо рядок тексту програми містить ідентифікатори if, для яких є відповідний ідентифікатор else;
- лічильник ребер збільшується на одиницю, якщо рядок тексту програми містить ідентифікатор else або else if;
- лічильник ребер збільшується на два, якщо рядок тексту програми містить ідентифікатори if, для яких відсутні відповідні ідентифікатори else та else if;
- лічильник ребер збільшується на два, якщо рядок тексту програми містить ідентифікатор циклу – for або while;
- лічильник ребер збільшується на N, якщо рядок тексту програми містить ідентифікатор галуження – switch.

Крок 4. Формування матриці інцидентності. Цей крок передбачає створення та заповнення матриці. Розміри матриці були визначені при виконанні попередніх кроків. Для виконання цього кроку створюються чотири списки:

List1 – містить номер вершини, якій відповідає рядок тексту програми.

List2 – містить номер вершини, яка є наступною після ідентифікатора галуження або ідентифікатора циклу.

List3 – містить маску, що відповідає типу рядка та може набувати таких значень:

- *functionName* – рядок, що містить ім’я функції;
- *returnLine* – рядок, що містить ідентифікатор return;
- *followLine* – рядок, що містить виклик іншої функції або присвоєння значення у змінну;
- *lastLineInsideIF* – останній рядок вихідного коду під конструкцією if;
- *lastLineInsideCycle* – останній рядок вихідного коду під конструкцією for або while;
- *lastLineInsideCase* – останній рядок вихідного коду під конструкцією case;

List4 – містить номер вершини ідентифікатора if, switch, for або while, під якою знаходиться рядок тексту програми.

Після створення та заповнення списків створюємо змінну *edgeC*, що буде використано для оновлення інформації у матриці, яке матриці відбувається згідно з такими правилами:

- якщо елемент *List1* містить 0 і елемент *List3* містить маску *functionName*, встановлюємо у комірці [0,0] значення «1», що відповідає виходу ребра з вершини 0.
- якщо елемент *List3* містить маску *followLine*, встановлюємо значення «-1» у комірці [*List1*[n], *edgeC*], збільшуємо лічильник вершин на одиницю та встановлюємо значення «1» у комірці [*List1*[n], *edgeC*];
- якщо елемент *List3* містить маску *lastLineInsideIF* встановлюємо значення «-1», у комірці [*List1*[n], *edgeC*], збільшуємо лічильник вершин на 1 та встановлюємо значення «1», у комірці [*List1*[n], *edgeC*]. У комірці [*List2*[n], *edgeC*] встановлюємо значення «-1» та встановлюємо значення «1», у комірці [*List4*[n], *edgeC*];
- якщо елемент *List3* містить маску *lastLineInsideCycle* встановлюємо значення «-1», у комірці [*List1*[n], *edgeC*], збільшуємо лічильник вершин на 1 та встановлюємо значення «1» у комірці [*List1*[n], *edgeC*], встановлюємо значення «-1» у комірці [*List4*[n], *edgeC*] і збільшуємо лічильник вершин на 1, встановлюємо значення «1» у комірці [*List4*[n], *edgeC*];
- якщо елемент *List3* містить маску *lastLineInsideCase* встановлюємо значення «-1» у комірці [*List1*[n], *edgeC*], збільшуємо лічильник вершин на 1 та встановлюємо значення «1» у комірці [*List1*[n], *edgeC*]. У комірці [*List2*[n], *edgeC*] встановлюємо значення «-1», а також встановлюємо значення «1» у комірці [*List4*[n], *edgeC*];
- якщо елемент *List3* містить маску *returnLine* встановлюємо значення «-1» у комірці [*List1*[n], *edgeC*];

Сформована матриця зберігається у окремому текстовому файлі, для подальшого її опрацювання.

4.2 Аналіз відповідності тексту програми і асемблерного коду. Цей аналіз дає змогу встановити перелік асемблерних інструкцій, що необхідні для виконання певного фрагменту коду, а також залежність тривалості їх виконання від внутрішніх модулів мікроконтролера.

Аналіз асемблерного коду дає змогу визначити кількість тактів, що необхідні для їхнього виконання. Вхідними даними для розрахунку тривалості виконання програмного коду є Listing-файл, що генерується у процесі компілювання програмного коду середовищем розроблення Keil µVision (рис. 3.3).

```
  int main(void)
b508            PUSH     {r3,lr}
  {
    HAL_Init();
f7fffffe        BL       HAL_Init

    /* Configure the system clock to 168 MHz */
    SystemClock_Config();
f7fffffe        BL       SystemClock_Config

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_GPIOD, ENABLE);
bf00            NOP
2000            MOVS     r0,#0
9000            STR      r0,[sp,#0]
4821            LDR      r0,|L3.152|
6800            LDR      r0,[r0,#0]
f0400008        ORR      r0,r0,#8
491f            LDR      r1,|L3.152|
6008            STR      r0,[r1,#0]
4608            MOV      r0,r1
6800            LDR      r0,[r0,#0]
f0000008        AND      r0,r0,#8
9000            STR      r0,[sp,#0]
bf00            NOP
bf00            NOP
```

Рисунок 3.3 – Фрагмент Listing-файлу, що відображає відповідність між текстом програми мовою С та асемблерним кодом

Загальна тривалість виконання функції обчислюється за формулою:

$$t_f = \sum_{i=0}^{m} (n_i \cdot t_i) + tr_i, \qquad (3.6)$$

де: m – кількість асемблерних інструкцій у функції; n_i – кількість тактів, за яку виконується асемблерна інструкція; t_i – тривалість виконання однієї обчислювальної операції, або тривалість надсилання одного байту інформації через інтерфейс зв'язку, що розраховується згідно з 3.3 – 3.5; tr_i – тривалість читання інструкції з пам'яті програм.

3.3 Метод формування плану тестування на підставі моделі функціонування ПЗ з урахуванням поведінки периферійних пристроїв

План тестування – це документ, що містить перелік та порядок функцій, які підлягають тестуванню [80]. Наявність плану тестування та його якість є передумовою, для проведення швидкого та ефективного тестування тривалості виконання програмного коду. Адже будь-яке сучасне спеціалізоване ПЗ вбудованих систем містить тисячі програмних функцій, що водночас містять сотні тисяч можливих шляхів виконання програми. Окрім цього, отримане значення тривалості виконання гілки програми після одного вимірювання не, може, трактуватись, як найбільша тривалість виконання програми. Тому для визначення найбільшої тривалості виконання програми потрібно значно збільшити кількість проведених вимірювань, що неможливо для сучасного ПЗ. Саме тому потрібно визначати перелік функцій та гілок, що підлягають більш ретельному тестуванню та не витрачати час та зусилля на тестування "незначних" функцій.

Метод формування плану тестування передбачає виконання наступних етапів:

Етап 1. Цей етап передбачає виконання синтаксичного аналізу всіх файлів, що входять до складу проекту, який розробляється в середовищі розробки Keil µVision. Аналіз map-файлу, створеного в процесі компіляції проекту дає змогу отримати перелік усіх функцій проекту, що вносяться у спеціальну таблицю в базі даних. Структура наведена у табл. 3.1.

Таблиця 3.1 – Таблиця для зберігання основних результатів оцінювання ТВПК

Ім’я функції	Номер гілки	Мін. пост. трив, с	Макс. пост. трив, с	Середня трив, с	Середня трив. – дисперсія, с	Середня трив. + дисперсі, с
Main	Гілка 1	$3.676 \cdot 10^{-7}$	$4.063 \cdot 10^{-7}$	0	0	0
FlashDataRead	Гілка 1	$1.244 \cdot 10^{-7}$	$1.375 \cdot 10^{-7}$	0	0	0
FlashDataRead	Гілка 2	$1.548 \cdot 10^{-7}$	$1.623 \cdot 10^{-7}$	0	0	0
EraseSector	Гілка 1	$1.866 \cdot 10^{-7}$	$2.063 \cdot 10^{-7}$	0.721	0.691	0.751
EraseSector	Гілка 1	$1.72 \cdot 10^{-7}$	$1.83 \cdot 10^{-7}$	0	0	0
EraseFlash	Гілка 1	$1.696 \cdot 10^{-3}$	$1.875 \cdot 10^{-3}$	0	0	0
EraseFlash	Гілка 2	$1.866 \cdot 10^{-7}$	$2.063 \cdot 10^{-7}$	5.452	1.386	9.518

Етап 2. Розділяємо усі програмні інструкції на дві групи, тривалість виконання яких не залежить від периферійних пристроїв і залежить від них. Всі інструкції першої групи, що написані з використанням мови високого рівня, можуть бути представлені у вигляді асемблерних інструкцій. Тривалість виконання цих інструкцій залежить від архітектури мікроконтролера, зокрема у мікроконтролерах з архітектурою RISC кожна команда виконується за один такт, в CISC інструкція, може, виконуватись упродовж 1-12 тактових імпульсів. Використовуючи можливості середовища розроблення Keil µVision, а саме аналізуючи Listing-файл встановлюємо відповідність програмних та асемблерних інструкцій (рис. 3.3). На цьому етапі обчислюємо загальну тривалість всіх інструкцій високого рівня, що містяться всередині функції, що оцінюється. Для цього алгоритм почергово обирає функцію з табл. 3.1, та ітеративно для кожного імені функції здійснює пошук тіла функції у файлах проекту. Тіло будь-якої функції починається з інструкції PUSH і завершується інструкцією POP у асамблерно-подібному коді. Цей факт використовується для розпізнавання першої та останньої інструкції у асемблерному коді. Для подальшого розрахунку ТВПК створюємо окрему таблицю у базі даних, у яку вносимо перелік усіх інструкцій, що підтримуються мікроконтролером, та кількість тактів, яка необхідна для її виконання (табл. 3.2).

Таблиця 3.2 – Таблиця для зберігання переліку асемблерних інструкцій

Асемблерна інструкція	Кількість тактів, що необхідна для її виконання
MOVE	1
ADD	1
ADDS	1
SUB	1
LDR	2

В результаті цього етапу до табл. 3.1 додається два значення для кожної функції – мінімальна постійна тривалість, що залежить від мікроконтролера та, відповідно, максимальна тривалість. Необхідність збереження двох значень замість одного пов'язана з тим, що тактова частота мікроконтролера є нестабільною та залежить від обраного генератора (кварцові генератори є найбільш точними, тоді як схеми RC взагалі поступаються їм у точності). Отже, мінімальна постійна ТВПК – результат підрахунку загальної тривалості всіх відповідних інструкцій асемблера *T*, що містить відхилення тактової частоти N%, тоді як максимальна постійна ТВПК дорівнює *T* + *N*%.

Етап 3. Встановлення апаратно-залежних інструкцій. Ідея цього етапу полягає у визначенні фрагментів програми тривалість яких залежить від часу відгуку апаратних компонентів. Для цього проводимо синтаксичний аналіз усіх файлів проекту, щоб встановити фрагменти програми, тривалість виконання яких залежить від часу відгуку периферійних пристроїв. Прикладом такого фрагменту є:

while (<waiting_for_hardware_response>) {};

Наведений фрагмент програмного коду відповідає за очікування зміни стану прапорця значення якого змінюється відповідно до стану апаратного компоненту. Зчитування стану компоненти, може, здійснюватися різними способами, наприклад: шляхом опитування відповідного виводу периферійного пристрою, в обробнику переривань, або через періодичне опитування регістра стану.

Кожен прапорець відображає стан лише одного апаратного компоненту, що значно полегшує збір та аналіз цих даних. У табл. 3.3 наведено приклад таблиці бази даних, що містить вхідну інформацію про кожен прапорець.

Таблиця 3.3 – Таблиця для зберігання переліку асемблерних інструкцій

Назва файлу	Назва функції	Номер рядку	Назва прапорця
Init	HardwareInit	15	SPI_I2S_FLAG_TXE
Init	HardwareInit	45	SPI_I2S_FLAG_RXNE
Init	HardwareInit	45	I2C_ FLAG_RDY
Emergency	Emergency_Task	18	UART_FLAG_RXNE
Emergency	Emergency_Task	29	UART_FLAG_RXNE
Emergency	Emergency_Task	31	UART_FLAG_TXNE
MainLoop	MainTask	45	SPI_I2S_FLAG_RXNE
Background	BackgrnTask	22	DMA_IT_TCIF0
Background	BackgrnTask	43	DMA_IT_TEIF0

Наведена таблиця буде використана у наступних етапах, зокрема під час дослідження діапазону зміни часу відгуку периферійного пристрою. Для цього на основі табл. 3.3 формуємо нову табл. 3.4 що зв’язує кожен прапорець з відповідним компонентом вбудованої системи, а також функцією, що його використовує.

Таблиця 3.4 – Таблиця для списку прапорців

Назва прапорця	Назва периферійного пристрою/Тип операції	Процедура
SPI_I2S_FLAG_TXE	AT45DB041D/ Запис буфера	Interrupt data send
SPI_I2S_FLAG_RXNE	AT45DB041D/ Зчитування буфера	Interrupt data receive
DMA_IT_TCIF0	Внутрішній ЦАП / Send data	Interrupt transfer complete
DMA_IT_TEIF0	Внутрішній ЦАП / Send data error	Interrupt transfer error

Етап 4. Оцінювання зміни часу відгуку периферійного пристрою та ТВПК. Припустимо, що функція, яка досліджується містить М-інструкції з певною невизначеністю тривалості виконання, які описуються інтервалами [a1, a2], [b1, b2] ... [x1, x2] (що наведено на рис. 3.4). Припускаємо, що тривалість виконання будь-якої апаратно-залежної операції відбувається за розподілом

Гауса та проводимо перевірку цього припущення на реальних вбудованих системах. Отримані результати близькі до середнього значення інтервалу, що наведений у документації на периферійний пристрій.

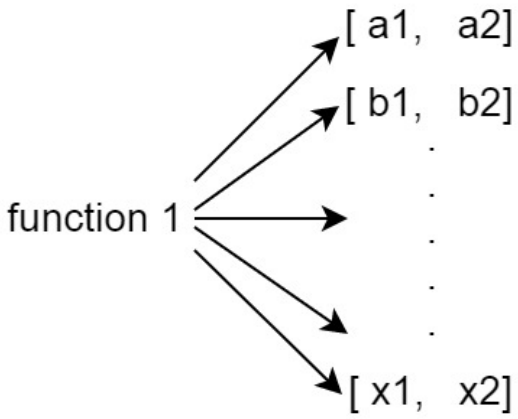


Рисунок 3.4 – Перелік інтервалів, що представляють розкид часу відгуку периферійних пристроїв

Для подальшого оцінювання слід знов розглянути "з нуля" кожну функцію. Для цього на основі табл. 3.1 створюємо нову таблицю (табл. 3.5). Ця таблиця повинна містити інформацію про інтервал зміни часу виконання тої чи іншої операції.

Таблиця 3.5 – Таблиця, що відображає час відгуку периферійних пристроїв

Периферійний пристрій	Тип операції	Мін. час відгуку, с	Макс. Час відгуку, с
LIS302_DL	Read register	$20 \cdot 10^{-6}$	$20 \cdot 10^{-6}$
LIS302_DL	Read status	$20 \cdot 10^{-6}$	$20 \cdot 10^{-6}$
LIS302_DL	Write mode register	$20 \cdot 10^{-6}$	$20 \cdot 10^{-6}$
MT29F8G08FABWP	Read ID	$3 \cdot 10^{-6}$	$50 \cdot 10^{-6}$
MT29F8G08FABWP	Page read	$3 \cdot 10^{-6}$	$700 \cdot 10^{-6}$
MT29F8G08FABWP	Program page	$300 \cdot 10^{-6}$	$700 \cdot 10^{-6}$
MT29F8G08FABWP	Block erase	$2 \cdot 10^{-3}$	$3 \cdot 10^{-3}$
LIS3DH	Read register	$10 \cdot 10^{-6}$	$25 \cdot 10^{-6}$
LIS3DH	Read status	$30 \cdot 10^{-6}$	$50 \cdot 10^{-6}$
LIS3DH	Write mode register	$30 \cdot 10^{-6}$	$50 \cdot 10^{-6}$
AT45DB041D	Page erase	$13 \cdot 10^{-3}$	$32 \cdot 10^{-3}$
AT45DB041D	Page erase	$13 \cdot 10^{-3}$	$32 \cdot 10^{-3}$
AT45DB041D	Block erase	$30 \cdot 10^{-3}$	$75 \cdot 10^{-3}$

На основі табл. 3.5 проводимо дослідження зміни тривалості виконання програмного коду. Для цього алгоритм ітеративно генерує N множини випадкових значень, нормально розподілених всередині інтервалів [a1, a2], [b1, b2] ... [x1, x2] і в кожній ітерації обчислює ТВПК функції (за допомогою наведеного вище методу статичного аналізу). Середнє значення, що отримане у внаслідок проведених чисельних експериментів характеризує найбільш імовірне значення ТВПК. А значення дисперсії (3.7) вказує максимальне значення, за яким тривалість виконання ξ у будь-якому окремому експерименті, може, відрізнятися від середнього значення.

$$D\xi \approx \frac{1}{N-1}\left[\sum_{j=1}^{N}\xi^2 - \frac{1}{N}\left(\sum_{j=1}^{N}\xi_j\right)^2\right], \qquad (3.7)$$

У табл. 3.1 додаємо розраховані значення: середня ТВПК – дисперсія та середня ТВПК + дисперсія. На практиці можливий варіант коли інструкції, що мають певну невизначеність у їх тривалості виконання розміщуються у декількох гілках однієї функції. Отже, існує необхідність пов'язувати кожну інструкцію з випадковим відхиленням тривалості її виконання, а також гілкою функції, у який вона викликається. Цей підхід дасть змогу окремо оцінити ТВПК для кожної гілки, і таким чином визначити ті гілки функції, які потребують більш ретельного тестування.

Етап 5. Розраховуємо вагу кожної функції. Повторно здійснюємо синтаксичний аналіз тексту програми та встановлюємо перелік потоків, що створюються. Для кожного з цих потоків визначаємо таку інформацію: пріоритет, періодичність запуску, імовірність створення потоку, та його ваговий коефіцієнт. Ця інформація вноситься у табл. 3.6.

Таблиця 3.6 – Таблиця, що містить інформацію про потоки програми

ІД Потоку	Інформація про потік			
	Пріоритет	Періодичність, с	Ймовірність створення	Ваговий коефіцієнт
tid_Hardware_Init_Task	1	1	1	5
tid_Software_Init_Task	1	1	1	5
tid_Main_Task	0	1	1	4
tid_Communication_Task	2	$10 \cdot 10^{-3}$	1	600
tid_Background_Task	1	$20 \cdot 10^{-3}$	1	250
tid_Acquisition_Task	2	$100 \cdot 10^{-3}$	1	60
tid_Emergency_Task	3	10000	0.5	0.00035

Визначення пріоритету потоку передбачає необхідність пошуку виклику усіх екземплярів функції osPrioritySet і osThreadCreate, що розміщенні у проекті. Аналізуючи параметри, що передаються у ці функції ми отримуємо пріоритет для кожного потоку. Крім цього, якщо в ході виконання програми змінюється пріоритет потоку на деякий час, який неможливо встановити використовуючи статичний аналіз програмного коду, то для подальшого аналізу обираємо найвище встановлене значення пріоритету.

Аналізуємо життєвий цикл кожного потоку для визначення періодичності виконання кожного потоку та ймовірності його створення. Для цього аналізуємо умови за яких викликається функції osThreadCreate та osThreadTerminate. Якщо функція osThreadCreate викликається завжди, без застосування умови її виконання то ймовірність створення такого потоку рівна 1. Однак якщо функція osThreadCreate викликається за певної умови то інженер тестувальник повинен вказати ймовірність її створення, що є в межах від 0 до 1. Синтаксичний аналіз обробника переривань від системного таймера дає змогу встановити періодичність запуску потоку. У разі, якщо потік є не періодичним, тобто після завершення передбаченої дії він видаляється тоді значення періодичності встановлюємо рівним 1.

Розрахунок вагового коефіцієнту потоку здійснюємо згідно 3.8 та вносимо результат у табл. 3.6.

$$w_{coef} = (priority + 4) \cdot \frac{P_{creation}}{periodicity}, \quad (3.8)$$

де: $priority$ – пріоритет потоку в якому викликається досліджувана функція; $periodicity$ – період повторного виклику потоку в якому викликається досліджувана функція; $P_{creation}$ – ймовірність створення потоку в якому викликається досліджувана функція. У цій формулі ми використовуємо зсув на 4 тому, що найнижче значення пріоритету у операційній системі CMSIS-RTOS RTX дорівнює -3, а найвище 3.

Останнім етапом є розрахунок ваги кожної функції згідно 3.9.

$$w_{function} = \frac{T_{max} - T_{min}}{2} \cdot w_{coef}, \quad (3.9)$$

де T_{max}, T_{min} – мінімальна та максимальна тривалість виконання функції, що розрахована на попередніх етапах.

На основі отриманих результатів створюємо ще одну таблицю, що буде містити наступну інформацію: ім’я функції, ІД потоку у якому вона викликається, а також її ваговий коефіцієнт (табл. 3.7).

Таблиця 3.7 – Таблиця, що відображає важливість тестування кожної функції

Назва функції	ІД потоку	Вага функції
SaveDataOnFlash	1	300
ReadAddOfCloudServer	2	275
SaveBackupDataOnFlash	1	240
ReadBackupDataFromFlash	1	240
ReadDataFromFlash	1	112

На підставі запропонованого методу формування плану тестування, що ґрунтується на моделі функціонування ПЗ і враховує поведінку периферійних пристроїв, розроблено архітектуру програмного модуля (рис. 3.5) та алгоритму його роботи (рис. 3.6).

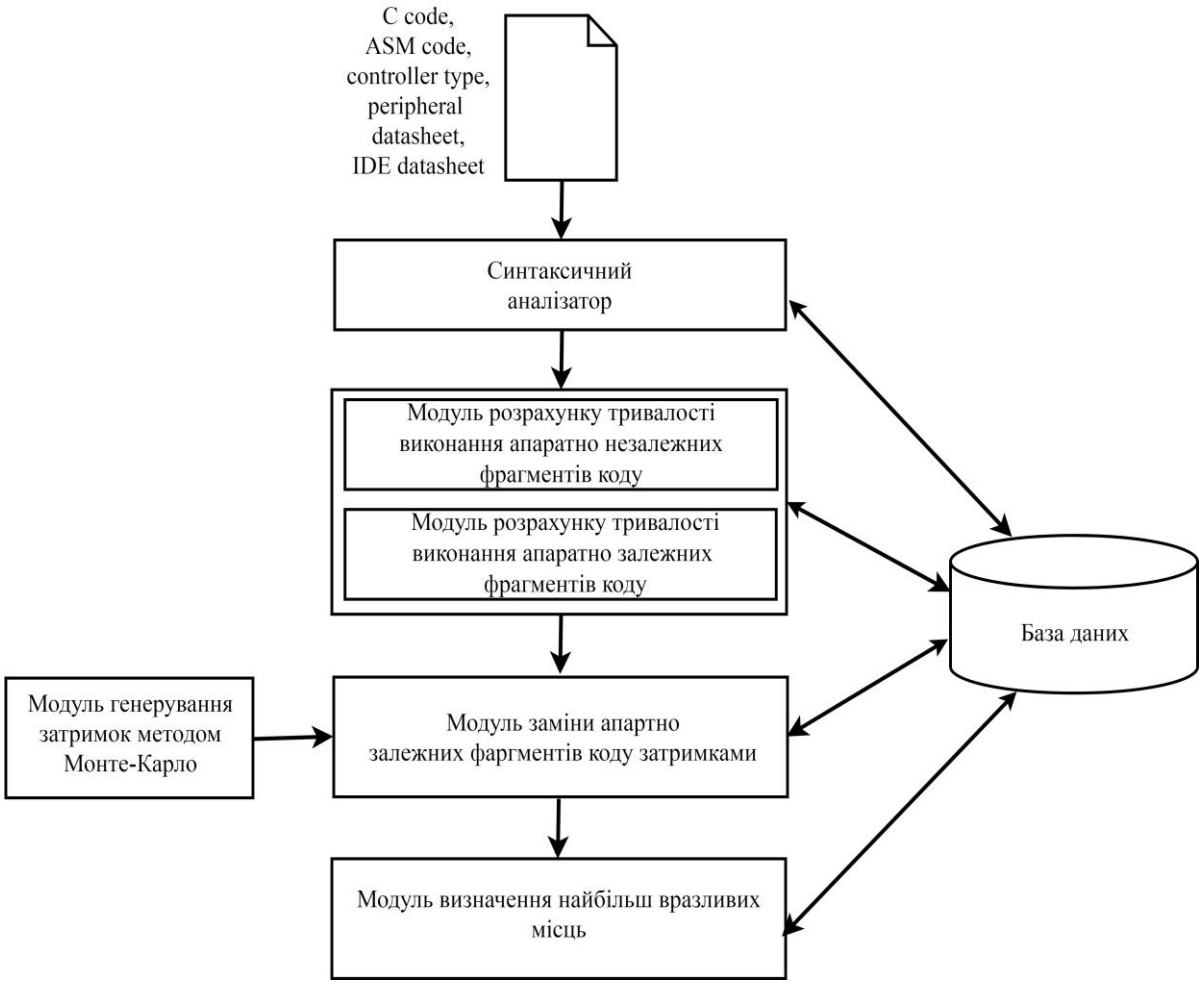


Рисунок 3.5 – Архітектура програмної компоненти для генерування плану тестування

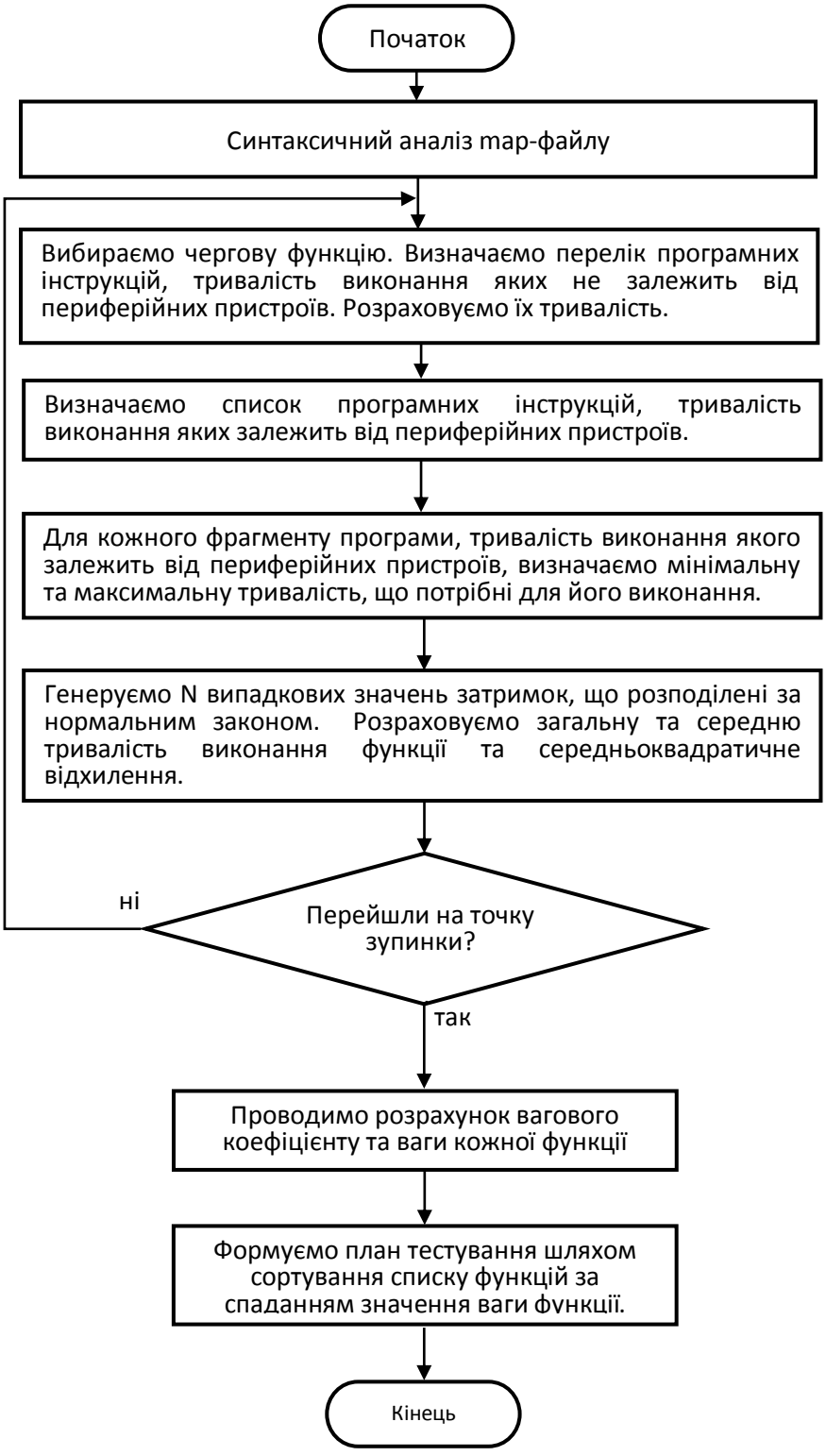


Рисунок 3.6 – Схема алгоритму, що реалізовує метод формування плану тестування на підставі моделі функціонування ПЗ з урахуванням поведінки периферійних пристроїв

3.3 Оцінювання складності алгоритму формування плану тестування за кількістю обчислювальних операцій

Оцінювання складності розробленого алгоритму формування плану тестування на підставі моделі функціонування ПЗ запропоновано проводити шляхом обчислення кількості обчислювальних операцій, що потрібно виконати для його реалізації.

Запропонований алгоритм формування плану тестування передбачає послідовне виконання декількох етапів тому його обчислювальна складність є сумою значень обчислювальної складності кожного етапу, а саме:

$$O_{\text{М}} = O_{\text{СА}} + O_{\text{ЕРТВК}} + O_{\text{ГЗ}} + O_{\text{РПТВ}} + O_{\text{РВК}} + O_{\text{ФПТ}}, \quad (3.10)$$

де: $O_{\text{СА}}$ – обчислювальна складність етапу синтаксичного аналізу; $O_{\text{ЕРТВК}}$ – обчислювальна складність етапу розрахунку тривалості виконання фрагментів коду з константною тривалістю виконання (апаратно незалежних); $O_{\text{ГЗ}}$ – обчислювальна складність етапу генерування змінних методом Монте-Карло; $O_{\text{РПТВ}}$ – обчислювальна складність етапу розрахунку показників тривалості виконання програмного коду; $O_{\text{РВК}}$ – обчислювальна складність етапу розрахунку вагових коефіцієнтів; $O_{\text{ФПТ}}$ – обчислювальна складність етапу формування плану тестування.

Етап синтаксичного аналізу передбачає виконання пошуку усіх функцій проекту, а також визначення фрагментів програми тривалість, яких залежить не залежить від апаратного забезпечення. Обчислювальна складність цього етапу може бути розрахована згідно з (3.11):

$$O_{\text{СА}} = O_{\text{ФСФ}}(1) + O_{\text{ФСФПзКТ}}(N), \quad (3.11)$$

де: $O_{\text{ФСФ}}$ – обчислювальна складність формування списку функцій; $O_{\text{ФСФПзКТ}}$ – обчислювальна складність процесу формування списку програмних інструкцій з константною тривалістю виконання; N – кількість функцій у проекті.

Другий етап передбачає здійснення розрахунку тривалості виконання програмного коду, що не залежить від часу відгуку периферійних пристроїв. Обчислювальна складність цього етапу залежить від кількості рядків тексту

програми та кількості асемблерних інструкцій, що характеризуються різною тривалістю виконання отже, визначається як $O_{\text{ЕРТВК}}(M^2)$, де M – кількість асемблерних функцій.

Обчислювальна складність третього етапу, що передбачає потребу генерування змінних методом Монте-Карло, залежить від кількості функцій програми, кількості гілок у цих функціях, а також кількості змінних. Обчислювальна складність цього етапу визначається, як $O_{\text{ГЗ}}(K^2)$, де K – кількість змінних, що потрібно згенерувати.

Етап розрахунку показників тривалості виконання програмного коду є найбільш трудомістким, адже передбачає проведення розрахунку загальної і середньої тривалості виконання програмного коду, а також середньоквадратичного відхилення. Обчислювальна складність цього етапу розраховується за формулою:

$$O_{\text{РПТВ}} = O_{\text{РЗТВФ}}(N) + O_{\text{РСТВФ}}(N) + O_{\text{РСКВ}}(N), \quad (3.12)$$

де: $O_{\text{РЗТВФ}}$– обчислювальна складність процесу розрахунку загальної тривалості виконання програмного коду; $O_{\text{РСТВФ}}$ – обчислювальна складність процесу розрахунку середньої тривалості виконання програмного коду; $O_{\text{РСКВ}}$ – обчислювальна складність процесу розрахунку середньоквадратичне відхилення; N – кількість функцій у проекті.

Обчислювальна складність етапу розрахунку вагових коефіцієнтів залежить тільки від кількості функцій проекту, а відтак визначається, як $O_{\text{РВК}}(N)$, де N – кількості функцій проекту.

Етап формування плану тестування полягає у сортуванні списку функцій за значенням вагового коефіцієнта, складність якого залежить від кількості функцій проекту – $O_{\text{ФПТ}}(N)$, де N – кількості функцій проекту.

З урахуванням 3.11, 3.12, а також формули 3.10, загальна складність алгоритму буде визначатися за формулою:

$$O_{\text{М}} = O_{\text{ФСФ}}(1) + O_{\text{ФСФПзКТ}}(N) + O_{\text{ЕРТВК}}(M^2) + O_{\text{ГЗ}}(K^2) + O_{\text{РЗТВФ}}(N) + O_{\text{РСТВФ}}(N) + O_{\text{РСКВ}}(N) + O_{\text{РВК}}(N) + O_{\text{ФПТ}}(N) = O(M^2). \quad (3.13)$$

Згідно з проведеним розрахунком складності алгоритму розраховуємо час, що потрібний для генерування плану тестування проектів з загальною кількістю функцій 50, 100 та 200. Розрахунок буде виконаний з використання наступних припущень:

- комп'ютер, що виконує алгоритм для формування плану тестування виконує 1,1 млрд. операцій в секунду.
- кожна функція містить у собі, щонайменше 200 асемблерних операцій.

Результати розрахунку наведені у табл. 3.8.

Таблиця 3.8 – Час формування плану тестування для різної кількості функцій проекту

К-ть функцій проекту	Загальна к-ть асемблерних операцій, тис	Час формування плану тестування, с
50	10	0,1
100	20	0,4
200	40	1,6

Отримані результати розрахунку обчислювальної складності алгоритму формування плану тестування характеризуються не високою обчислювальною складністю метода, а також високою швидкістю його виконання. Результати розрахунку часу роботи алгоритму, свідчать про придатність результатів та доцільність реалізації алгоритму для проведення експериментальних досліджень.

3.4 Висновки до розділу

1) Отримав подальший розвиток метод статичного аналізу тривалості виконання програмного коду вбудованих систем, який враховує внутрішню архітектуру мікроконтролерів. Цей метод удосконалено, шляхом врахування втрати швидкості під час обміну даними між внутрішніми модулями мікроконтролера, а також при надсиланні даних через інтерфейси зв’язку. Окрім цього, запропонований метод підлягає повній автоматизації, а відтак дає змогу інтегрувати його у будь-яку систему автоматизованого тестування.

2) Отримала подальший розвиток модель функціонування ПЗ, в якій на відміну від наявних моделей враховується поведінка периферійних пристроїв,

шляхом врахування змінної тривалості виконання операцій, що дає змогу підвищити ступінь адекватності моделей функціонування ПЗ вбудованих систем жорсткого РЧ.

3) Запропоновано метод формування плану тестування, що використовує запропоновану модель функціонування ПЗ, в якому на відміну від наявних методів враховано вагу кожної функції, що дає змогу підвищити ефективність тестування тривалості виконання програмного коду вбудованих систем жорсткого РЧ.

4) Розроблено архітектуру та алгоритм роботи програмного модуля, що реалізує запропонований метод формування плану тестування, який враховує поведінки периферійних пристроїв.

5) Оцінювання обчислювальної складності розробленого алгоритму свідчить про придатність отриманих результатів, доцільність його реалізації та використання у програмному засобі для автоматизованого тестування тривалості виконання програмного коду, що наведений у розділі 5. Зменшення часу витраченого на побудову плану тестування можливе завдяки використанню паралельних обчислень, а саме паралельного виконання розрахунку тривалості виконання фрагментів програми.

6) Подальше удосконалення розробленого методу формування плану тестування можливе за рахунок врахування ймовірності виклику функцій в залежності від дій користувача вбудованої системи, а також завдяки урахування впливу на вбудовану системи зовнішніх та внутрішніх чинників, а саме: температури навколишнього середовища, напруги живлення, та деградації її компонентів у зв’язку з їх старінням.

Основні результати та висновки розділу опубліковано в працях [39, 20, 100].

РОЗДІЛ 4. ПРОГНОЗУВАННЯ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ З УРАХУВАННЯМ ВПЛИВУ ЗОВНІШНІХ ТА ВНУТРІШНІХ ЧИННИКІВ

У процесі розроблення моделей функціонування ПЗ, що застосовуються для аналізу ТВПК, використовують тільки ті характеристики вбудованої системи, що описують її стан. Характеристики, що описують вплив зовнішніх чинників на вбудовану систему, такі як температура навколишнього середовища, напруга живлення, атмосферний тиск та інші прийнято не брати до уваги, позаяк це дає змогу:

- спростити процес розроблення моделі, а відтак зменшити витрачений час на її створення;
- скоротити витрати часу на аналіз моделей реального ПЗ;
- відсутні експериментальні дані, які містять інформацію про залежність тривалості виконання програми від температури, напруги живлення тощо.

Саме тому наявні моделі функціонування ПЗ характеризуються низьким рівнем адекватності. Удосконалення моделей функціонування ПЗ шляхом урахування впливу на вбудовану систему: температури навколишнього середовища, напруги живлення, атмосферного тиску чи деградації компонентів завдяки старінню, підвищить рівень їх адекватності, що дасть змогу використовувати ці моделі для:

- перевірки відповідності категорії розміщення вбудованих систем [93];
- перевірки відповідності вимогам кліматичного виконання;
- перевірки відповідності вимогам та їх уточненням до електроживлення;
- визначення максимального періоду експлуатації вбудованих систем і періоду технічного обслуговування для забезпечення вимог тривалості виконання критичних, з точки зору безпечності, фрагментів коду;

В цьому розділі подано метод аналізу ТВПК, що дає змогу врахувати вплив зовнішніх та внутрішніх чинників, а також здійснити її прогнозування.

4.1 Вплив зовнішніх та внутрішніх чинників на тривалість виконання програмного коду

З початком дослідження тривалості виконання програмного коду, теоретики та практики цієї галузі під час обговорення використовували таке визначення: це час, що потрібний для виконання певного фрагменту коду в заданому контексті на заданому апаратному забезпеченні. При цьому вплив апаратного забезпечення на ТВПК вони розглядали через врахування архітектури мікроконтролера (мікропроцесора). Однак, ці моделі апаратного забезпечення для спрощення процесу розроблення та аналізу використовують такі припущення:

- ТВПК не змінюється без еволюції програми;
- ТВПК не змінюється, якщо ПЗ не експлуатується.

Наведені припущення є схожими до тих, що використовувались під час розроблення моделей надійності ПЗ, які не враховували вплив апаратного забезпечення, та характеризувалися низьким рівнем адекватності та точності отриманих результатів. Тому, врахування впливу зовнішніх і внутрішніх чинників, що діють на компоненти вбудованої системи протягом тривалого часу, та призводять до деградації їх параметрів і, як наслідок, до збільшення ТВПК на цьому апаратному забезпеченні.

Опираючись на інформацію, отриману зі звітів тестування апаратного забезпечення, можемо визначити перелік чинників, що призводять до чималої деградації компонентів вбудованої системи, а саме:

- *Температура навколишнього середовища* є домінуючим стрес-фактором, адже впливає на усі компоненти вбудованої системи. Довготривалий вплив високої температури навколишнього середовища не тільки підвищує ТВПК, а й підвищує ймовірність відмови апаратного забезпечення. Окрім цього, поєднання впливу температури з іншими стрес-факторами призводить до підсилення процесу деградації компонентів. Саме тому температура навколишнього середовища є ключовим чинником, що потрібно враховувати під час аналізу ТВПК.

- *Тиск*. Вбудовані системи або їхні деякі фрагменти можуть виконувати роботу в умовах підвищеного тиску, що призводить до їх деградації. Крім цього, робота в умовах циклічної зміни тиску, що діє на апаратне забезпечення, призводить до підвищення рівня деградації апаратного забезпечення та зростання ТВПК.
- *Вологість*. Відносна вологість повітря, у якому працює вбудована система, може, змінюватись в межах від 10 до 100%, у поєднанні з високою температурою наносить руйнівний вплив для корпусу вбудованої системи, а з часом і компонентів вбудованої системи. В процесі руйнування (деградації) компонентів спостерігається погіршення параметрів елементів та збільшення тривалості виконання програмного коду.
- *Вібрація*. Вібрація, що формується поблизу вбудованої системи передається через корпус на її компоненти. Компоненти, що руйнуються з часом, призводять до збільшення значення тривалості виконання програмного коду.

На підставі визначеного переліку чинників, які призводять до деградації компонентів, можемо сформувати перелік стрес факторів, що можуть розглядатися при аналізі тривалості виконання програмного коду:

1) Температура:
    - висока температура навколишнього середовища;
    - різка зміна температури навколишнього середовища;
2) Тиск:
    - високий тиск навколишнього середовища;
    - різка зміна тиску у навколишньому середовищі;
3) Вологість:
    - висока вологість навколишнього середовища;
    - висока вологість в середині корпус вбудованої системи;
4) Вібрація:
    - механічна вібрація під час роботи;

- механічна вібрація під час транспортування вбудованої системи;

5) Напруга живлення:
   - висока вхідна напруга;
   - низька вхідна напруга;
   - коливання напруги живлення;
6) Інше:
   - радіація;
   - старіння;
   - хімічний склад навколишнього середовища

Отже, потрібно розробити метод аналізу тривалості виконання програмного коду, що враховує вплив зовнішніх і внутрішніх чинників на апаратне забезпечення вбудованих систем, тим самим забезпечить підвищення достовірності отриманих результатів.

4.2 Метод аналізу тривалості виконання програмного коду з урахуванням впливу зовнішніх і внутрішніх чинників на апаратне забезпечення вбудованих систем

Для аналізу тривалості виконання фрагменту програми подаємо її у вигляді графу потоку керування (рис. 4.1).

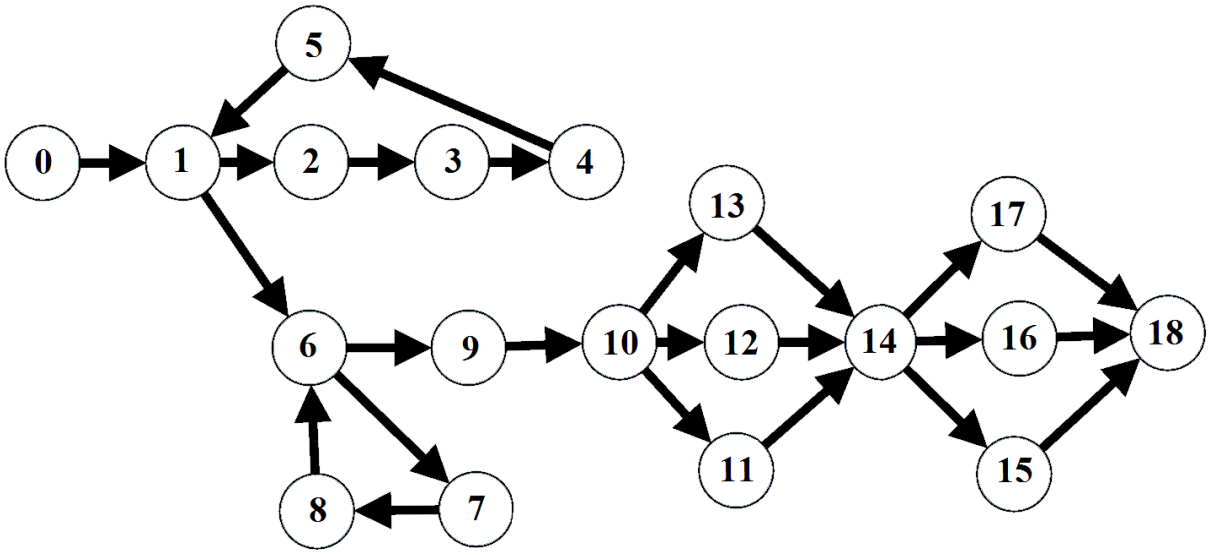


Рисунок 4.1 – Приклад графу потоку керування, що репрезентує функцію програми

Тривалість виконання певної гілки програми прийнято розраховувати за формулою:

$$\tau_b = \sum_{i=1}^{n} \tau_i, \qquad (4.1)$$

де: τ_i – тривалість виконання інструкції програмного коду, що наведено у вигляді вершини графу; n – загальна кількість вершин у гілці програми.

З огляду на те, що ТВПК залежить не тільки від кількості обчислювальних операцій, а й від дії на апаратне забезпечення тих чи інших чинників, формула (4.1) набуває такого вигляду:

$$\tau_b = \sum_{j=1}^{k} \tau_j + \sum_{i=1}^{m} \tau_i(param), \qquad (4.2)$$

де: τ_j – тривалість виконання програмної інструкції, що характеризується постійним значенням тривалості виконання; k – кількість програмних інструкцій з константною тривалістю виконання; τ_i – тривалість виконання програмної інструкції, яка залежить від часу відгуку периферійного пристрою та змінюється під дією на нього певного чинника $param$; m – кількість програмних інструкцій тривалість виконання яких залежить від часу відгуку периферійного пристрою.

Отже, на підставі моделі функціонування ПЗ, що наведена у вигляді графу потоку керування та формули (4.2), яка дає змогу врахувати залежність зміну часу відгуку периферійних пристроїв під час дії на неї зовнішніх та внутрішніх чинників, розроблено метод аналізу ТВПК [40, 38], що складається з таких етапів:

1. *Синтаксичний аналіз тексту програми, що водночас має ряд підзадач:*
   - 1.1. *Визначення фрагментів програми, тривалість виконання яких залежить від часу відгуку периферійних пристроїв*

Визначення фрагментів програми, тривалість яких залежить від часу відгуку периферійних пристроїв, є достатньо складним завданням в практичній реалізації. Одним із способів вирішення даної проблеми є використання

складних синтаксичних аналізаторів тексту програми, що проводять пошук таких синтаксичних конструкцій, тривалість виконання яких є невизначеною через поведінку периферійних пристроїв, або інтерфейсів зв’язку. Наприклад:

```
while(<waiting_for_hardware_response>) {};
```

Ще одним варіантом визначення фрагментів програми, є "ручний" аналіз, що проводиться інженером тестувальником. Використання такого підходу передбачає наявність детального розуміння принципів функціонування вбудованої системи, зв’язку мікроконтролера та периферійних пристроїв, а також розуміння принципів роботи інтерфейсів зв’язку. Такий підхід є більш часозатратним. Окрім цього, якість отриманого результату повністю залежить від кваліфікації та досвіду інженера, що проводить синтаксичний аналіз.

1.2. Розрахунок тривалості виконання фрагментів програми з константною тривалістю.

Усі фрагменти програми, що не увійшли до переліку фрагментів програми, тривалість яких залежить від часу відгуку периферійних пристроїв, автоматично відносяться до фрагментів програми з константною тривалістю виконання. Процес розрахунку їх тривалості полягає у синтаксичному аналізі листинг-файлу, що створюється середовищем розроблення Keil µVision під час компіляції проекту, і є поєднанням тексту програми написаних мовою С та асемблерних команд. Отже, загальна тривалість фрагменту програми є добутком кількості машинних тактів, що потрібні для виконання усіх інструкцій на тривалість виконання одного машинного такту. Визначення кількості тактів, що потрібні для виконання асемблерної інструкції, виконується на підставі даних від виробника мікроконтролера або згідно з інформацією про архітектуру мікроконтролера [26].

2. *Пошук та аналіз інформації про залежність часу відгуку периферійного пристрою від зовнішніх або внутрішніх чинників.*

Для визначених на попередньому етапі фрагментів програми, тривалість яких залежить від часу відгуку периферійних пристроїв, проводимо пошук інформації про залежність часу відгуку периферійного пристрою від зовнішніх

або внутрішніх чинників. Для цього інженери детально аналізують документацію на периферійний пристрій (даташит), проводять пошук звітів про тестування цих компонент (test report), або проводять експериментальні дослідження для визначення цих залежностей.

3. *Вибір математичної моделі апроксимації*

Експериментальні дані, що були отримані на попередньому етапі, використовуються при розрахунку коефіцієнтів апроксимації для декількох моделей апроксимації [96]. Для вибору найбільш відповідної моделі апроксимації, ми перевіряємо наскільки результати, отримані з використання тої чи іншої моделі, відповідають експериментальним даним. Для цього, проводимо візуальне порівняння залежностей, що побудовані за експериментальними даними та даними, що отримані під час розрахунку. Також проводимо розрахунок та порівняння значення середньої похибки апроксимації [92]. На підставі отриманих результатів приймаємо рішення про вибір моделі апроксимації.

4. *Прогнозування тривалості виконання програмного коду*

Використовуючи розраховані коефіцієнти апроксимації, для кожного фрагменту програми, тривалість якого залежить від часу відгуку периферійного пристрою, проводимо розрахунок тривалості виконання фрагменту програми при певному значенні фізичної величини, для якого відсутні експериментальні дані. Користуючись формулою (4.2), розраховуємо загальну тривалість виконання гілки програми з урахування впливу зовнішніх чи внутрішніх чинників.

На підставі розробленого методу реалізовано програмний модуль, який входить до складу засобу автоматизованого тестування вбудованих систем, описаний у розд. 5, що дає змогу аналізувати ТВПК вбудованих систем з урахуванням дії на нього зовнішніх чи внутрішніх чинників.

4.2 Дослідження ефективності запропонованого методу аналізу тривалості виконання програмного коду з урахуванням впливу зовнішніх та внутрішніх чинників

Для дослідження ефективності використання запропонованого методу, що враховує вплив зовнішніх та внутрішніх чинників на апаратне забезпечення вбудованої системи, проведено обчислення тривалості виконання програмного коду для п’яти вбудованих системи Італійської компанії Dinamica Generale [28]: Tomato Sniper (BC1), Smart Control (BC2), Pump Logger (BC3), Field Scale (BC4), DG8000-IC (BC5). У якості чинників, що діють на апаратне забезпечення вбудованої систем та змінюють ТВПК обрано: температуру навколишнього середовища, напругу живлення вбудованої системи та старіння компонентів. Вхідні дані про залежність часу відгуку від температури навколишнього середовища та напруги живлення отримані з документації на сигма-дельта АЦП ADS1118 [4]. У табл. 4.1 подані дані про залежність часу відгуку від температури навколишнього середовища при різних значеннях напруги живлення: 1) напруга живлення рівна 2В; 2) напруга живлення знаходиться в межах 3.3 - 5 В.

Таблиця 4.1 – Залежність часу відгуку периферійного пристрою від температури при різних значення напруги живлення

Температура навколишнього середовища, °С	Час відгуку при напрузі живлення (2В), с	Відхилення часу відгуку від номінального значення, %	Час відгуку при напрузі живлення (3,3/5В), с	Відхилення часу відгуку від номінального значення, %
-60	0,007626	-2,40	0,007890	1,00
-40	0,007648	-2,10	0,007875	0,70
-20	0,007671	-1,80	0,007867	0,60
0	0,007695	-1,50	0,007851	0,50
20	0,007710	-1,30	0,007846	0,40
40	0,007734	-1,00	0,007843	0,30
60	0,007746	-0,85	0,007843	0,30
80	0,007773	-0,50	0,007835	0,25
100	0,007789	-0,30	0,007835	0,25
120	0,007804	-0,10	0,007828	0,30
140	0,007820	0,10	0,007828	0,30

Згідно з даними, що подані у табл. 4.1 максимальне відхилення часу відгуку під температури становить 2,4%, однак величина відхилення залежить від конкретного елемента. Залежність часу відгуку від температури навколишнього середовища наведено на рисунках 4.2 - 4.3.

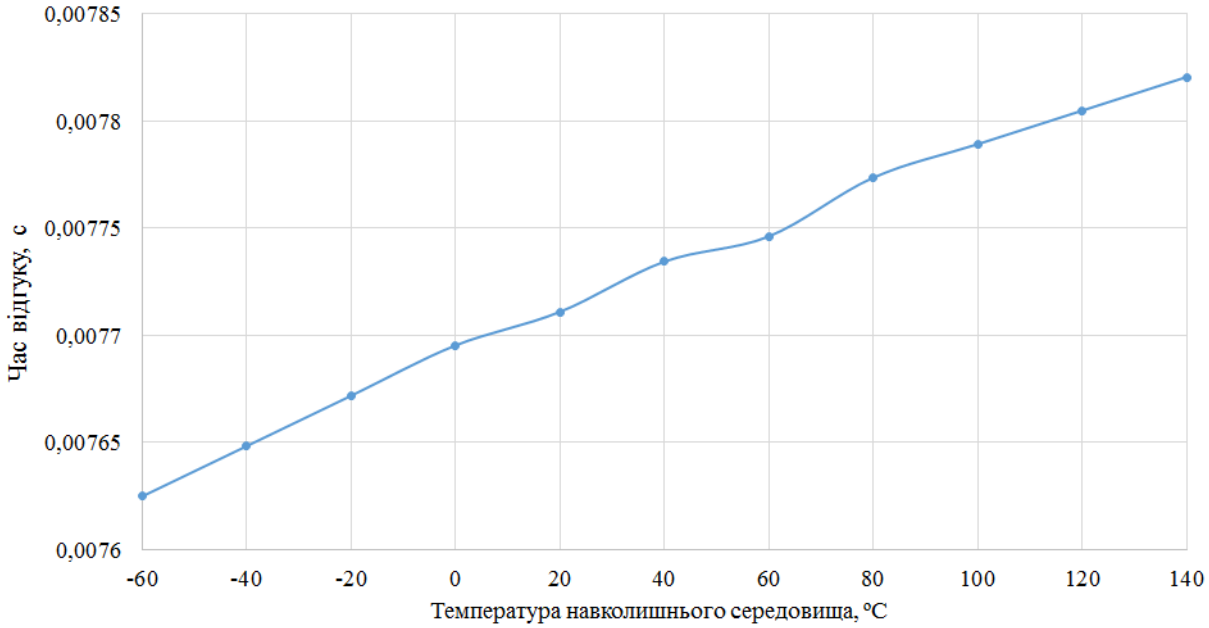


Рисунок. 4.2 – Залежність часу відгуку периферійних пристроїв від температури навколишнього середовища (напруга живлення 2 В) [4]

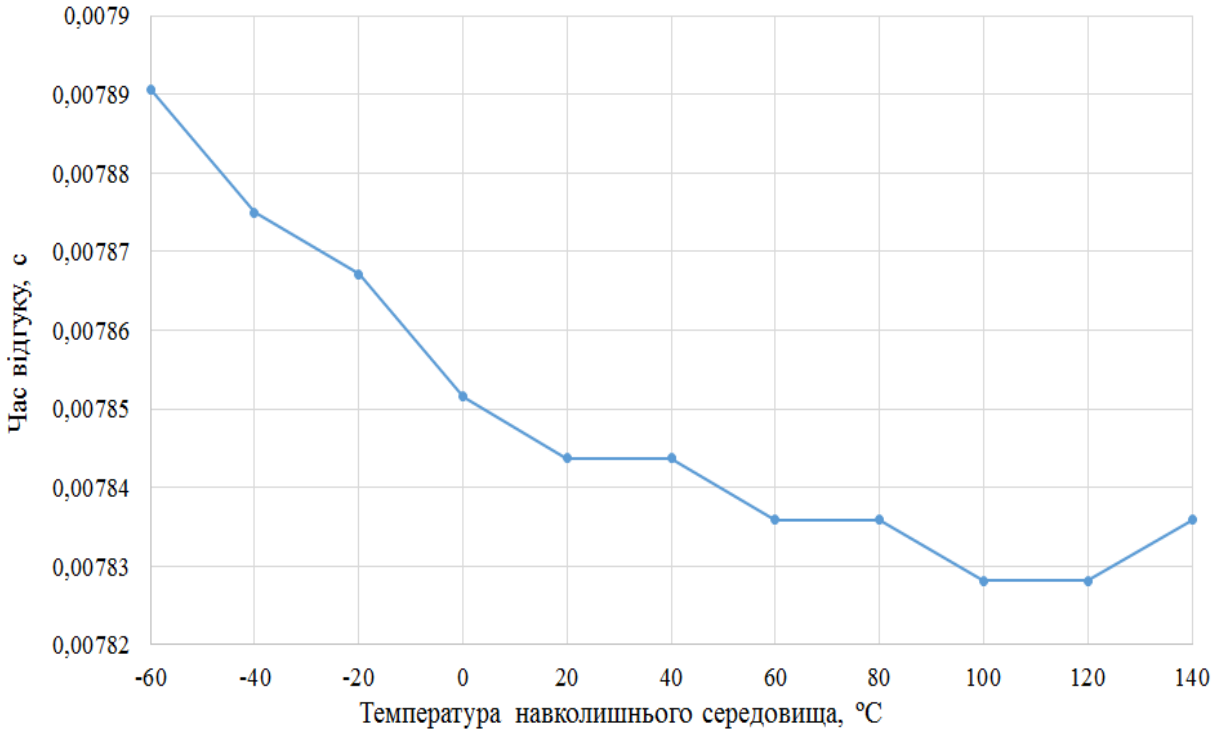


Рисунок. 4.3 – Залежність часу відгуку периферійних пристроїв від температури навколишнього середовища (напруга живлення 3,3-5 В) [4]

У табл. 4.2 наведено залежність часу відгуку АЦП ADS1118 від напруги живлення. Номінальний час відгуку (відповідно до офіційної табл. 8 [4]) становить 0,007812 с.

Таблиця 4.2 – Залежність часу відгуку від напруги живлення

Напруга живлення, В	Час відгуку, с	Відхилення часу відгуку від номінального значення, %
2,0	0,007718	-1,20
2,1	0,007718	-1,20
2,2	0,007726	-1,10
2,3	0,007734	-1,00
2,4	0,007734	-1,00
2,5	0,007742	-0,90
2,6	0,007757	-0,70
2,7	0,007773	-0,50
2,8	0,007789	-0,30
2,9	0,007804	-0,10
3,0	0,007812	0,0
3,1	0,007828	0,20
3,2	0,007835	0,30
3,3	0,007843	0,40
3,4	0,007844	0,41
3,5	0,007844	0,41
3,6	0,007847	0,45
4,0	0,007849	0,47
4,5	0,007850	0,48
5,0	0,007851	0,50

У табл. 4.3 наведені експериментальні дані [29] про залежність зміни часу відгуку периферійних пристроїв від періоду експлуатації (що зумовлено старінням компонентів).

Таблиця 4.3 – Залежність часу відгуку від періоду експлуатації (старіння)

Період експлуатації, роки	Час відгуку, с	Відхилення часу відгуку від номінального значення, %
0,25	0,008593	10
0,60	0,008593	10
1,70	0,008593	10
1,90	0,008984	15
2,20	0,008984	15
3,00	0,009218	18

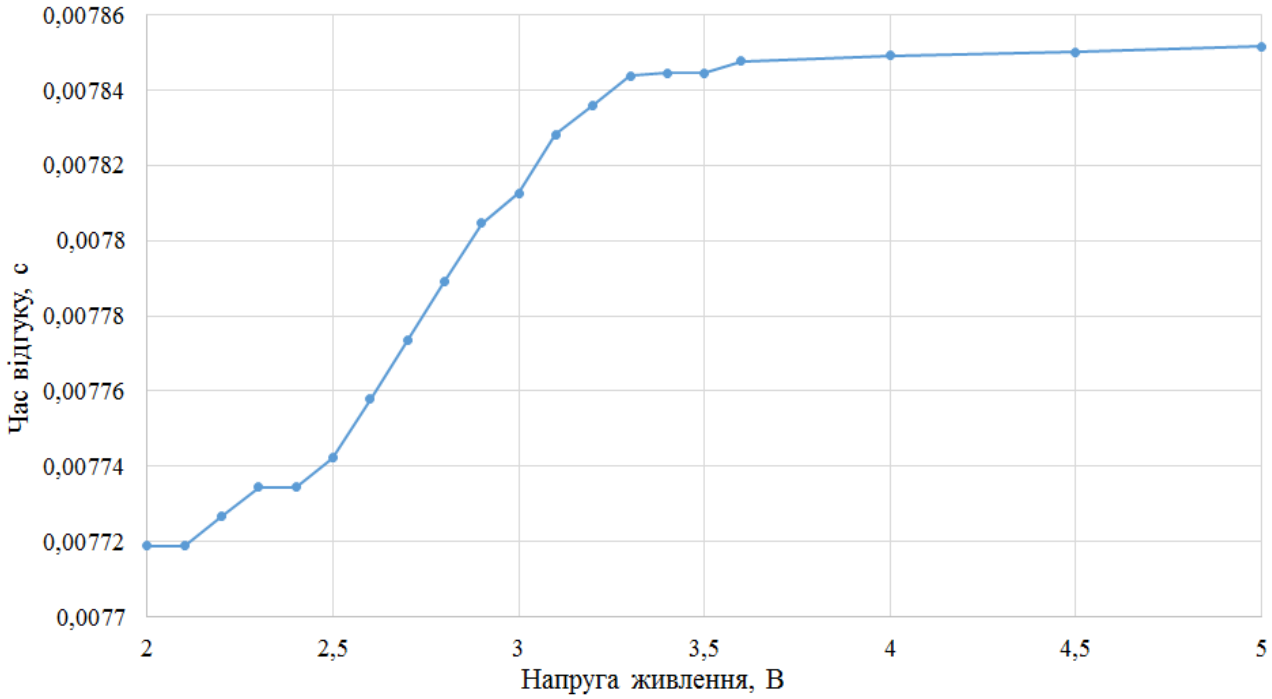


Рисунок. 4.4 – Залежність часу відгуку периферійних пристроїв від напруги живлення [4]

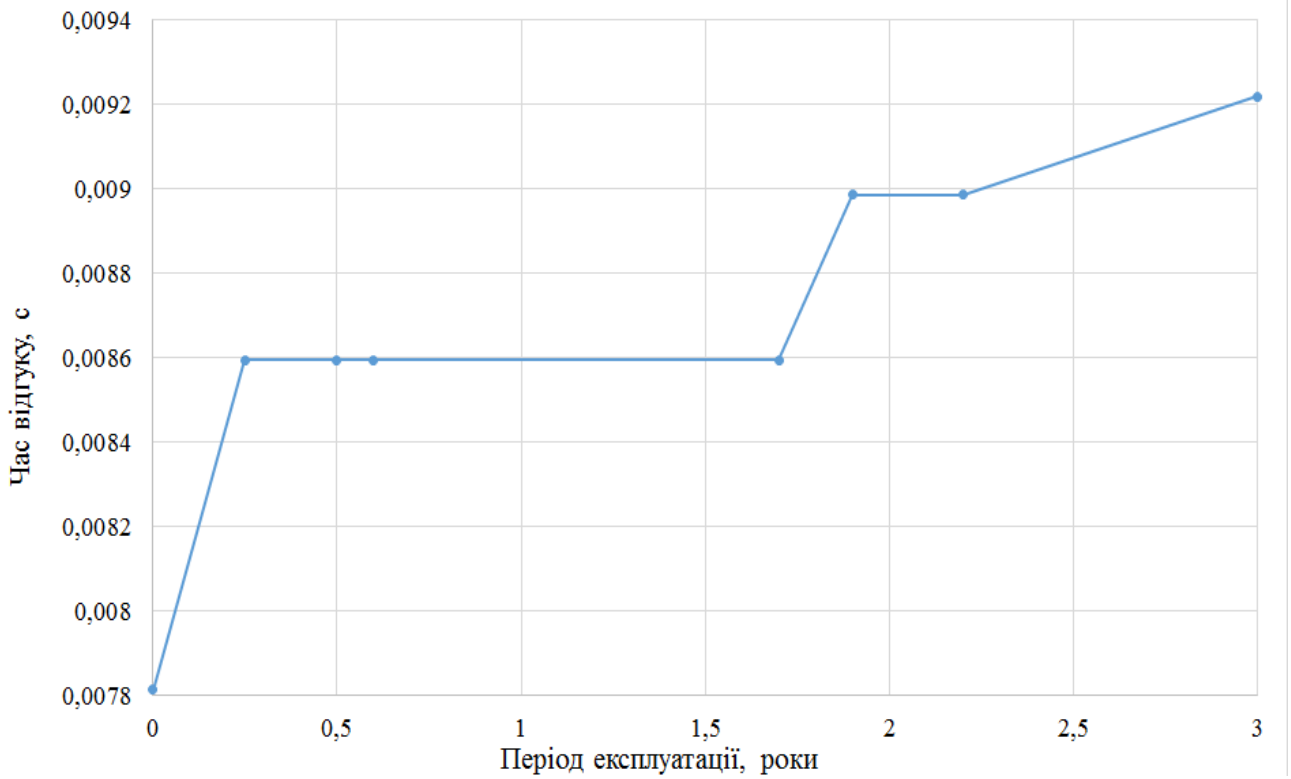


Рисунок. 4.5 – Залежність часу відгуку периферійних пристроїв від періоду експлуатації (старіння)[29]

На підставі поданих вхідних даних та використовуючи метод кубічної апроксимації проведено дослідження впливу фізичних величин на тривалість виконання фрагментів програми, що є критичними з точки зору безпечності.

Дослідження 1. В ході проведеного дослідження залежності тривалості виконання програмного коду від напруги живлення встановлено, що зниження напруги живлення периферійних пристроїв до 2В призводить до зростання тривалості виконання функцій в середньому на 2,2%. Подальше зниження напруги живлення периферійних пристроїв не призводить до збільшення часу їх відгуку, адже рівень напруги живлення <2В недостатній для функціонування периферійних пристроїв. Отримані результати наведені у табл. 4.4.

Таблиця 4.4 – Результати дослідження впливу напруги живлення на ТВПК

	ВС1	ВС2	ВС3	ВС4	ВС5
Тривалість виконання програмного код без урахування впливу напруги живлення, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного код з урахування впливу напруги живлення, мс	2,5	7,3	14,01	18,61	8,95
Відносна похибка, %	2,2	2,4	2,44	2,2	2,1

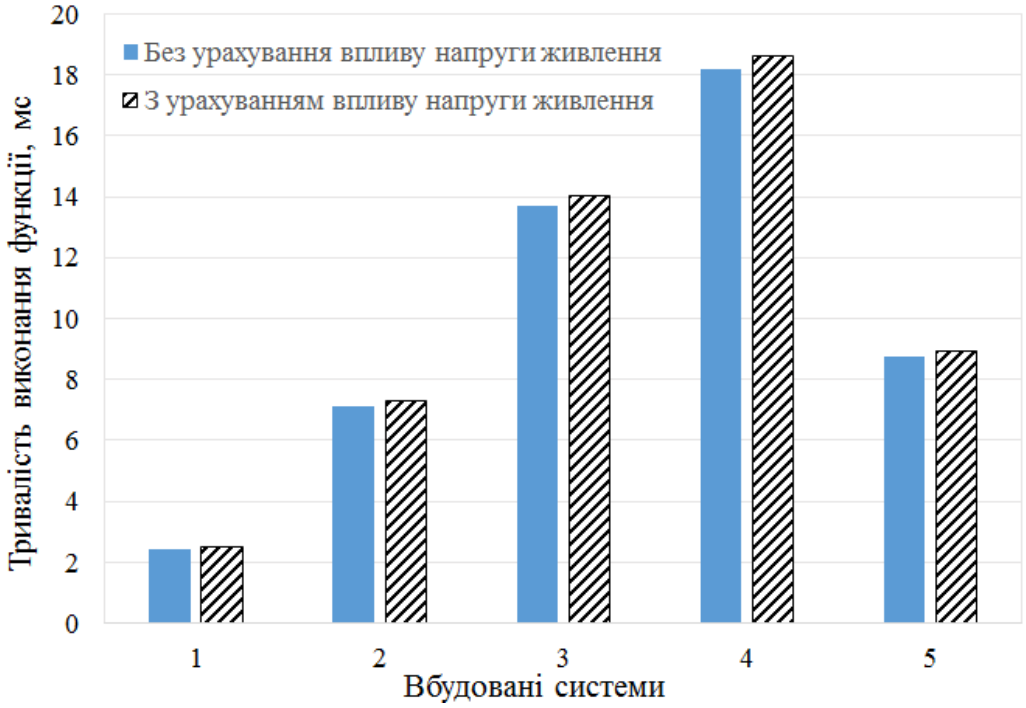


Рисунок 4.6 – Порівняння точності оцінювання тривалості виконання програмного коду з урахуванням та без урахування впливу напруги живлення

Висновки до дослідження 1: Аналіз отриманих результатів показав, що різниця між значеннями тривалості виконання програмного коду залежить від периферійних пристроїв вбудованої системи, а також кількості звертань до них у критичних, з точки зору безпечності, фрагментів програми. Ігнорування залежності тривалості виконання програмного коду від напруги живлення призводить до заниження отриманих результатів, а тому є обов'язковим при оцінюванні тривалості виконання програмного коду автономних вбудованих систем РЧ, що розробляються для роботи від джерел живлення з низьким рівнем напруги та потужності (ultra-low-power).

Дослідження 2. В ході дослідження залежності тривалості виконання програмного коду від температури навколишнього середовища виявлено, що збільшення температури навколишнього середовища на 50°С призводить до зростання часу відгуку периферійного пристрою та зростання тривалості виконання програми в середньому на 2,1%. Зменшення температури навколишнього середовища на 50°С призводить до зменшення часу відгуку периферійних пристроїв, що в свою чергу призводить до зменшення тривалості виконання програми в середньому на 2,5%. Залежність тривалості виконання програмного коду при різних значеннях температури навколишнього середовища, показана на рис. 4.5.

Таблиця 4.5 – Результати дослідження впливу температури навколишнього середовища на ТВПК ($t_{НС}$ = -50°С).

	ВС №1	ВС №2	ВС №3	ВС №4	ВС №5
Тривалість виконання програмного коду без урахування впливу температури навколишнього середовища, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного коду з урахування впливу температури навколишнього середовища, мс	2,49	7,27	13,96	18,59	8,96
Відносна похибка, %	2	2,1	2,1	2,1	2,2

Таблиця 4.6 – Результати дослідження впливу температури навколишнього середовища на ТВПК ($t_{НС}$ = 70°С).

	ВС1	ВС2	ВС3	ВС4	ВС5
Тривалість виконання програмного коду без урахування впливу температури навколишнього середовища, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного коду з урахування впливу температури навколишнього середовища, мс	2,49	7,27	13,96	18,59	8,96
Відносна похибка, %	2	2,1	2,1	2,1	2,2

Таблиця 4.7 – Результати дослідження впливу температури навколишнього середовища на ТВПК ($t_{НС}$ = 120°С).

	ВС1	ВС2	ВС3	ВС4	ВС5
Тривалість виконання програмного коду без урахування впливу температури навколишнього середовища, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного коду з урахування впливу температури навколишнього середовища, мс	2,49	7,27	13,96	18,59	8,96
Відносна похибка, %	2	2,1	2,1	2,1	2,2

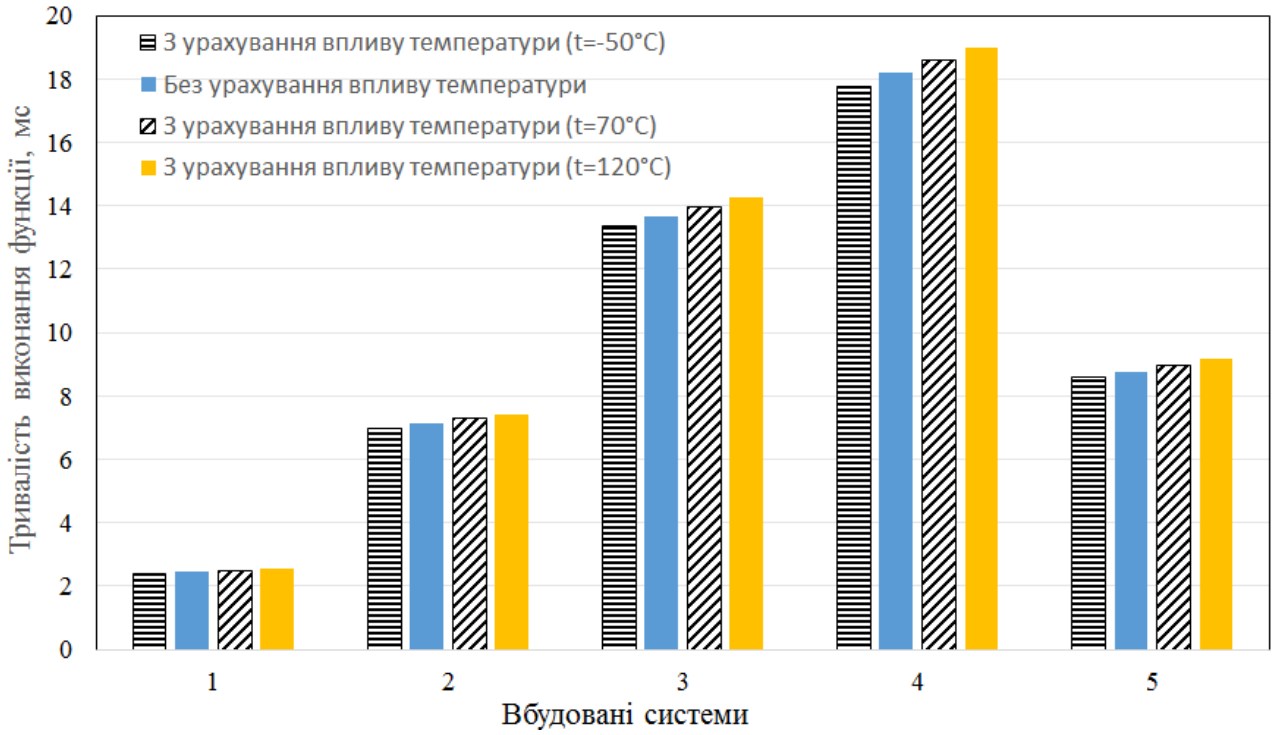


Рисунок 4.7 – Порівняння точності оцінювання тривалості виконання програмного коду з урахуванням та без урахування впливу температури навколишнього середовища

Висновки до дослідження 2: Аналіз отриманих результатів показав, що різниця між значеннями тривалості виконання програмного коду збільшується з ростом температури навколишнього середовища. Врахування впливу температури навколишнього середовища дасть змогу підвищити достовірність отриманих результатів та здійснити більш безпечне оцінювання тривалості виконання програмного коду. Варто відзначити, що врахування впливу температури навколишнього середовища потрібно обов'язково проводити у випадках, якщо вбудована система повинна працювати при високих температурах навколишнього середовища.

Дослідження 3. Дослідження зміни тривалості виконання програмного коду від періоду експлуатації (старіння компонентів). Це дослідження є складнішим, адже дані про залежність часу відгуку периферійних пристроїв від періоду їх експлуатації відсутні у стандартній документації. Тому при використанні метода прогнозування тривалості виконання програмного коду прийнято припущення, що тенденція зміни часу відгуку периферійних пристроїв досліджуваних вбудованих систем є аналогічною результатам опублікованим у звіті про тестування [100]. Залежність тривалості виконання програмного коду від періоду експлуатації (старіння компонентів), показана на рис. 4.8.

Таблиця 4.8 – Результати дослідження впливу періоду експлуатації вбудованих систем на ТВПК ($t_{ЕКСП}$ = 3 роки).

	ВС1	ВС2	ВС3	ВС4	ВС5
Тривалість виконання програмного коду без урахування впливу старіння периферійних пристроїв, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного коду з урахування впливу старіння периферійних пристроїв, мс	2,89	8,42	16,18	21,46	10,32
Відносна похибка, %	18,1	18,1	18,3	17,9	17,7

Таблиця 4.9 – Результати дослідження впливу температури навколишнього середовища на ТВПК ($t_{ЕКСП}$ = 5 років).

	ВС1	ВС2	ВС3	ВС4	ВС5
Тривалість виконання програмного коду без урахування впливу старіння периферійних пристроїв, мс	2,45	7,13	13,68	18,21	8,77
Тривалість виконання програмного коду з урахування впливу старіння периферійних пристроїв, мс	3,06	8,92	17,07	22,78	10,96
Відносна похибка, %	25	25,2	24,8	25,1	25

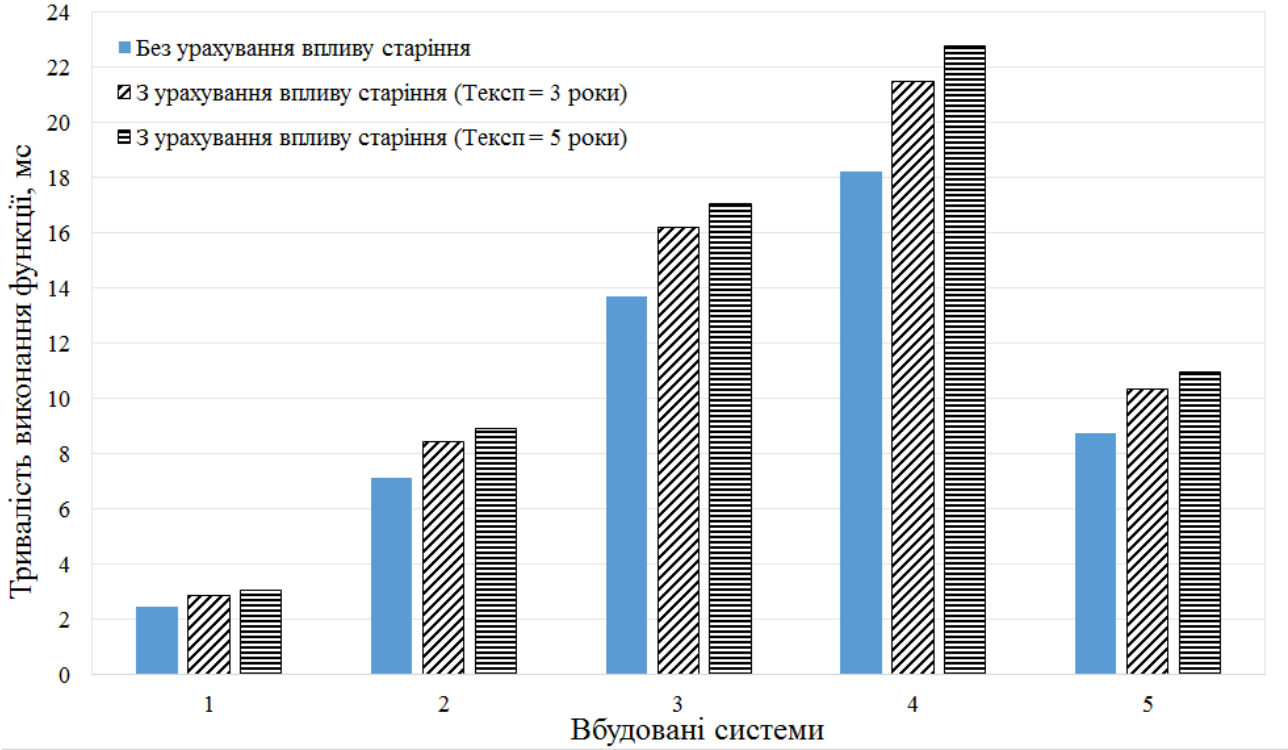


Рисунок 4.8 – Порівняння точності оцінювання тривалості виконання програмного коду з урахуванням та без урахування впливу старіння периферійних пристроїв

Висновки до дослідження 3: Аналіз отриманих результатів показав, що старіння компонентів у порівнянні з іншими зовнішніми та внутрішніми чинниками має найбільший вплив на ТВПК. Окрім цього, старіння компонентів притаманне для усіх вбудованих системи, а тому цей вид аналізу потрібно проводити для усіх вбудованих систем жорсткого РЧ.

4.3 Використання запропонованого методу для прогнозування тривалості виконання програмного коду

Окрім створення методу аналізу тривалості виконання програмного коду з урахуванням впливу зовнішніх чи внутрішніх чинників, є багато інших відкритих питань, пов’язаних з практичним використанням розробленого методу аналізу тривалості виконання, наприклад, прогнозування тривалості виконання програмного коду.

Прогнозуванням будемо називати деякий дослідницький процес в результаті якого ми отримуємо ймовірні дані про майбутній стан об’єкту для якого виконується прогнозування [99]. Отже, прогнозуванням тривалості виконання програмного коду будемо називати процес передбачення наступного значення тривалості виконання фрагменту програми на підставі аналізу його попереднього і поточного значення тривалості виконання фрагменту програми.

Прогнозування тривалості виконання програмного коду, зміна якої зумовлена старінням компонентів дає змогу: визначити момент часу коли ТВПК перевищує задане значення тривалості виконання програмного коду, встановити період технічного обслуговування системи, що забезпечить рівень тривалості виконання програмного коду в заданому діапазоні.

4.3.1 Прогнозування тривалості виконання програмного коду для визначення періоду технічного обслуговування вбудованої системи

Математичне прогнозування передбачає використання даних про деякі характеристики об’єкта, що прогнозується, обробку цих даних математичними методами, отримання залежностей, що пов’язують ці характеристики з часом, і розрахунку з допомогою знайденої залежності характеристик об’єкта в заданий момент часу. Весь процес прогнозування можна умовно розділити на такі етапи: збір та підготовка вхідних даних (статистики), вибір та обґрунтування математичної моделі об’єкта що прогнозується, обробка статистичних даних для визначення невідомих параметрів моделі і отримання залежностей, що зв’язують характеристики об’єкта з часом і рядом невідомих змінних,

прогнозування, тобто розрахунок значень параметрів об’єкта дослідження, що нас цікавлять в заданий момент часу.

Для прогнозування тривалості виконання програмного коду з використанням розробленого методу потрібно виконати лише останній етап, адже попередні етапи виконувались при проведенні аналізу тривалості виконання програмного коду.

Згідно з результатами опублікованими у [38] математична модель кубічної апроксимації забезпечує найменшу похибку апроксимації під час аналізу впливу старіння компонентів на ТВПК. Саме тому, математична модель кубічної апроксимації (4.3) буде застосовуватись для прогнозування тривалості виконання програмного коду.

$$\tau = aT^3 + bT^2 + cT + d \qquad (4.3)$$

де a, b, c і d коефіцієнти апроксимації, що розраховуються методом найменших квадратів.

Використовуючи формулу (4.2) проводимо прогнозування тривалості виконання програмного коду для періоду експлуатації 5 років. Результати прогнозування наведені в табл. 4.10. ТВПК без урахування старіння компонентів є константною та дорівнює 0,00611 мс.

Таблиця 4.10 – Результати прогнозування тривалості виконання програмного коду.

Період експлуатації вбудованої системи, роки	ТВПК з урахування старіння компонентів, с	Відносна похибка, %
0,0	0,00611	0,0
0,6	0,00643	5,20
1,1	0,00654	6,99
2,1	0,00653	7,21
2,6	0,00650	6,80
3,1	0,00652	6,77
3,6	0,00671	9,75
4,1	0,00713	16,72
4,6	0,00789	29,12
5,0	0,00907	48,45

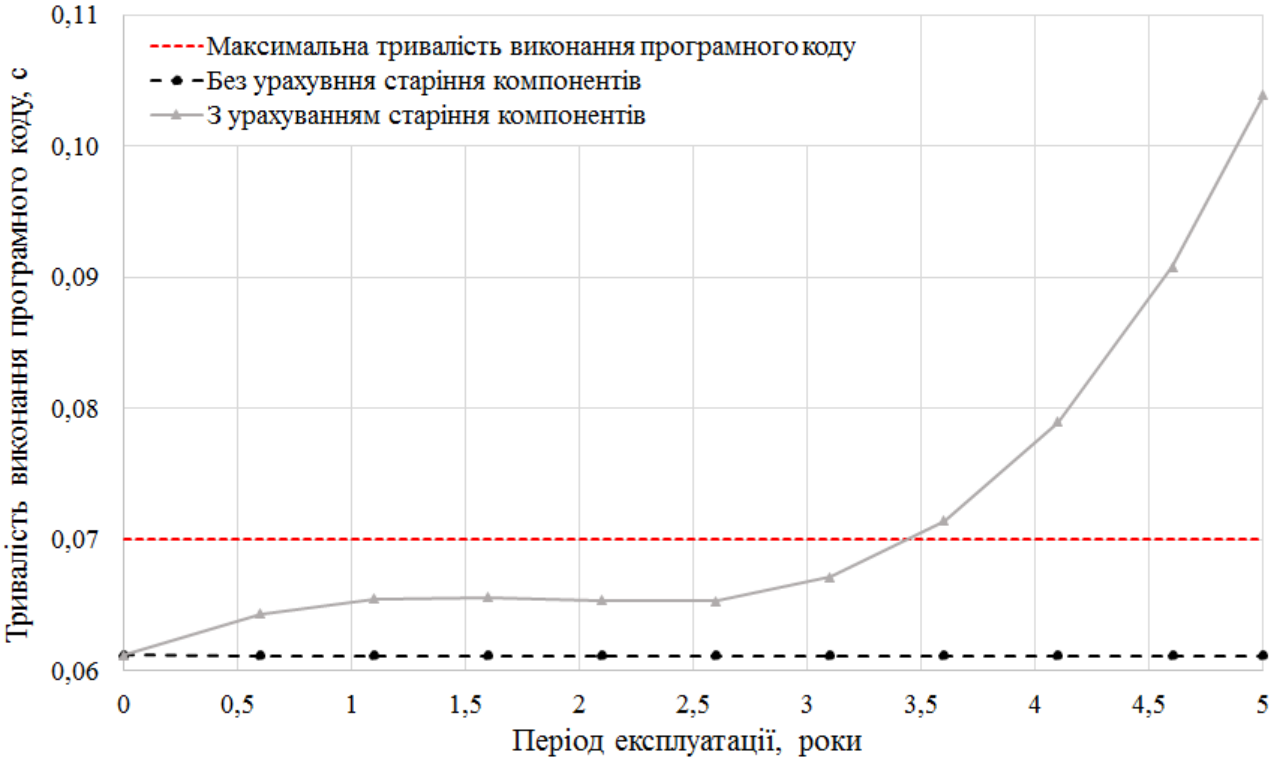


Рисунок 4.9 – ТВПК з урахування та без урахування старіння компонентів

Як видно з рис. 4.9 ТВПК перевищить максимально допустиме значення тривалості виконання програмного коду, що регламентовано технічним завданням, через 3,5 роки експлуатації. Саме тому, потрібно здійснити її технічне обслуговування (заміну компонентів, що призведуть до збільшення тривалості виконання програмного коду), або цю вбудовану систему необхідно вивести з експлуатації, адже вона є небезпечною, для життя та здоров'я людей, що з нею працюють.

4.3.2 Оцінювання точності моделі прогнозування

Цінність моделі прогнозування визначається її точністю, яка залежить від степені співпадіння майбутнього значення з оцінкою цього значення, що розраховано заздалегідь. Оскільки спостереження за реальними процесами здебільшого проводиться в умовах завад, а перебіг процесів виконується під дією різних випадкових величин, а відтак ми не можемо розраховувати на те, що прогноз майбутнього значення буде абсолютно точним.

Процес оцінювання точності моделі прогнозування можна розділити на такі етапи:

- розрахунок коефіцієнтів апроксимації для частини вхідних даних;
- прогнозування значень для відомих значень;
- оцінювання похибки прогнозу.

У якості вхідних даних для оцінювання точності моделі прогнозування використовуємо дані з табл. 4.11.

Таблиця 4.11 – Залежність часу відгуку від періоду експлуатації (старіння)

Період експлуатації, роки	Час відгуку, с	Відхилення часу відгуку від номінального значення, %
0,25	0,008593	10
0,60	0,008593	10
1,70	0,008593	10
1,90	0,008984	15

Розраховуємо коефіцієнти апроксимації *a, b, c* і *d.*

$$a = 0{,}0014; b = -0{,}004; c = 0{,}0032; d = 0{,}0079 \qquad (4.4)$$

Використовуючи розраховані коефіцієнти визначаємо час відгуку для періоду експлуатації 2,2 та 3 роки. Результати прогнозування та похибка прогнозування наведена в табл. 4.12.

Таблиця 4.12 – Залежність часу відгуку від періоду експлуатації (старіння)

Період експлуатації, роки	Час відгуку, с (експериментальні дані)	Час відгуку, с (дані отримані шляхом прогнозування)	Похибка прогнозування, %
2,2	0,008984	0,009143	16,72
3,0	0,009218	0,013310	44,27

За отриманими результатами оцінювання точності моделі прогнозування можемо зробити висновок, що модель прогнозування характеризується низьким рівнем точності, адже для розрахунку коефіцієнтів апроксимації використовувався малий обсяг вхідних даних. Збільшення обсягу

експериментальних даних залежності часу відгуку периферійних пристроїв від періоду експлуатації дасть змогу підвищити точність прогнозування.

4.4 Застосування методу аналізу тривалості виконання програмного коду на різних етапах життєвого циклу вбудованих систем

Розглянемо можливість застосування аналізу тривалості виконання програмного коду з урахуванням впливу зовнішніх та внутрішніх чинників на різних етапах життєвого циклу вбудованих систем. Для практичних потреб методу аналізу ТВПК повинен бути достатньо простим, а відтак модель на якій він базується не може описувати у деталях кожну особливість впливу зовнішніх чинників на вбудовану систему. Розглянемо розроблений метод стосовно його використання на етапах проектування, розроблення апаратного та програмного забезпечення, інтеграції програмного та апаратного забезпечення, та на етапі функціонування. Визначено етапи життєвого циклу, на яких використання розробленого методу є найбільш ефективним.

1. *Етап проектування.*

Застосування розробленого методу аналізу ТВПК на етапі проектування можливе після визначення переліку компонентів вбудованої систем, а також розроблення алгоритмів її функціонування. Оскільки, для аналізу ТПВК потрібен текст програми (який ще не існує) то у якості вхідних даних про ТВПК можемо використати тривалість виконання, що розрахована на основі обчислювальної складності алгоритму. Таким чином, цей метод дасть змогу оцінити конфігурацію апаратного забезпечення, а також відповідність алгоритму функціонування вбудованої системи її технічному завданню. Крім цього, цей метод дасть змогу встановити необхідну конфігурацію вбудованої системи, що забезпечить заданий рівень надійності та безпечності її експлуатації впродовж заданого періоду.

2. *Етап розроблення апаратного та програмного забезпечення.*

Особливістю цього етапу є те, що під час його виконання розроблене апаратне та програмне забезпечення тестується модульно. На цьому етапі перевіряється відповідність ПЗ до його функціональних вимог. Таким чином

застосування розробленого методу аналізу ТВПК, є недоцільним до завершення етапу інтеграційного тестування.

3. Інтеграція програмного та апаратного забезпечення.

На цьому етапі розроблене програмне забезпечення інтегрується у розроблене апаратне забезпечення та проводиться інтеграційне та системне тестування. Після успішного завершення функціонального тестування виконується нефункціональне тестування, зокрема тестування ТВПК. Використання розробленого методу на цьому етапі дасть змогу більш точно оцінити ТВПК для розробленого програмного та апаратного забезпечення, а також відповідність заданому у технічному завданні значенню надійності та безпечності.

4. Етап функціонування.

Коли програмне та апаратне забезпечення визнається повною відповідністю технічному завданню, вбудована система вводиться в експлуатацію. Враховуючи те, що етап функціонування включає у себе не лише експлуатацію вбудованої системи, а і технічне обслуговування апаратного забезпечення та оновлення програмного забезпечення, відтак застосування розробленого методу є не менш важливим, адже дає змогу оцінити відповідність вбудованої системи в процесі експлуатації, а також коректувати розроблений план технічного обслуговування у відповідності до останньої версії програмного забезпечення вбудованої системи.

4.5 Висновки до розділу

1) Запропоновано метод аналізу тривалості виконання програмного коду який враховує вплив зовнішніх чи внутрішніх чинників на апаратне забезпечення, що дає змогу забезпечити заданий рівень надійності та безпечності вбудованих систем.

2) Проведені дослідження методу аналізу тривалості виконання програмного коду підтвердило доцільність його застосування для усіх вбудованих систем жорсткого РЧ, зокрема для таких напрямів:

а) перевірки відповідності ТЗ до вбудованої системи, зокрема відповідності вимогам до категорії розміщення, кліматичного виконання та електроживлення;

б) використання запропонованого методу для прогнозування впливу зовнішніх і внутрішніх чинників на ТВПК, що дає змогу використовувати це метод для визначення періоду технічного обслуговування, чи максимального терміну експлуатації.

3) Запропонований метод є універсальним, що дає змогу враховувати не тільки вплив напруги живлення, температури чи старіння, а і інших зовнішніх чи внутрішніх чинників таких, як: атмосферний тиск, вологість повітря та інші.

4) Подальший розвиток запропонованого методу аналізу тривалості виконання програмного коду можливий за рахунок удосконалення математичної моделі, а саме врахування впливу одразу декількох незалежних зовнішніх чи внутрішніх чинників, що впливають на ТВПК. Це дасть змогу підвищити рівень адекватності отриманих результатів, а відтак підвищити рівень надійності та безпечності вбудованих систем РЧ.

Основні результати розділу 4 опубліковані в працях [38, 40].

РОЗДІЛ 5. АРХІТЕКТУРА ПРОГРАМНОГО ЗАСОБУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ ТА ЇЇ ПРАКТИЧНЕ ЗАСТОСУВАННЯ

Програмне забезпечення розроблене на основі методів, методики і алгоритмів поданих у розд. 2, 3 і 4 є новим ПЗ «EXTT», що призначений для:

- автоматизованого тестування тривалості виконання програмного коду;
- формування плану тестування з урахуванням поведінки периферійних пристроїв;
- прогнозування тривалості виконання програмного коду з урахуванням впливу на вбудовану систему зовнішніх чи внутрішніх фізичних величин.

Описана інформаційна модель розробленої програмної системи та основні кроки створення ПЗ. Наведено приклади використання розробленого ПЗ.

5.1 Архітектура програмного забезпечення для автоматизованого тестування тривалості виконання програмного коду

На підставі сформованої концепції системи, що описана у розд. 2, а також архітектури програмних модулів наведених у розд. 3-4, розроблена архітектура ПЗ «EXTT», яка графічно наведена на рис. 5.1.

Структурно, ПЗ «EXTT» складається з дев’яти функціонально завершених програмних компонентів:

- препроцесор;
- модуль розрахунку тривалості виконання функцій;
- модуль формування плану тестування з урахуванням поведінки периферійних пристроїв;
- модуль формування плану тестування без урахування поведінки периферійних пристроїв;
- модуль прогнозування тривалості виконання програмного коду з урахуванням впливу на вбудовану систему фізичних величин;

- модуль тестування тривалості виконання програмного коду;
- візуалізатор результатів тестування;
- віддаленого клієнта;
- бази даних.

Кожен з програмних компонентів є незалежним програмним модулем, що використовує мову XML для взаємодії між ними. Таке рішення робить можливим програмну реалізацію компонентів системи на різних мовах програмування та спрощує інтеграцію компонентів між собою. Взаємодія програмних модулів з базою даних здійснюють за допомогою SQL-запитів, що дає змогу використовувати в програмній системі практично будь-яку реляційну базу даних. Взаємодія між віддаленим клієнтом та модулем тестування виконується через мережу Інтернет з протоколом TCP/IP з використанням власного протоколу.

Характерною ознакою розробленої архітектури є потенційна працездатність в неповній конфігурації. З цієї точки зору не обов'язковими компонентами є модуль прогнозування тривалості виконання програмного коду, модуль генерування плану тестування з урахуванням поведінки периферійних пристроїв, модуль генерування плану тестування без урахування поведінки периферійних пристроїв. Для функціонування системи, як засобу автоматизованого тестування тривалості виконання програмного коду, достатня наявність модуля тестування, а також будь-якого одного з двох модулів генерування плану тестування. Подібна відсутність компонентів призведе до звуження функціональних можливостей системи, але її працездатність збережеться.

Нижче наводиться загальний огляд кожного з програмних компонентів системи і розкриваються властиві їм завдання та функції.

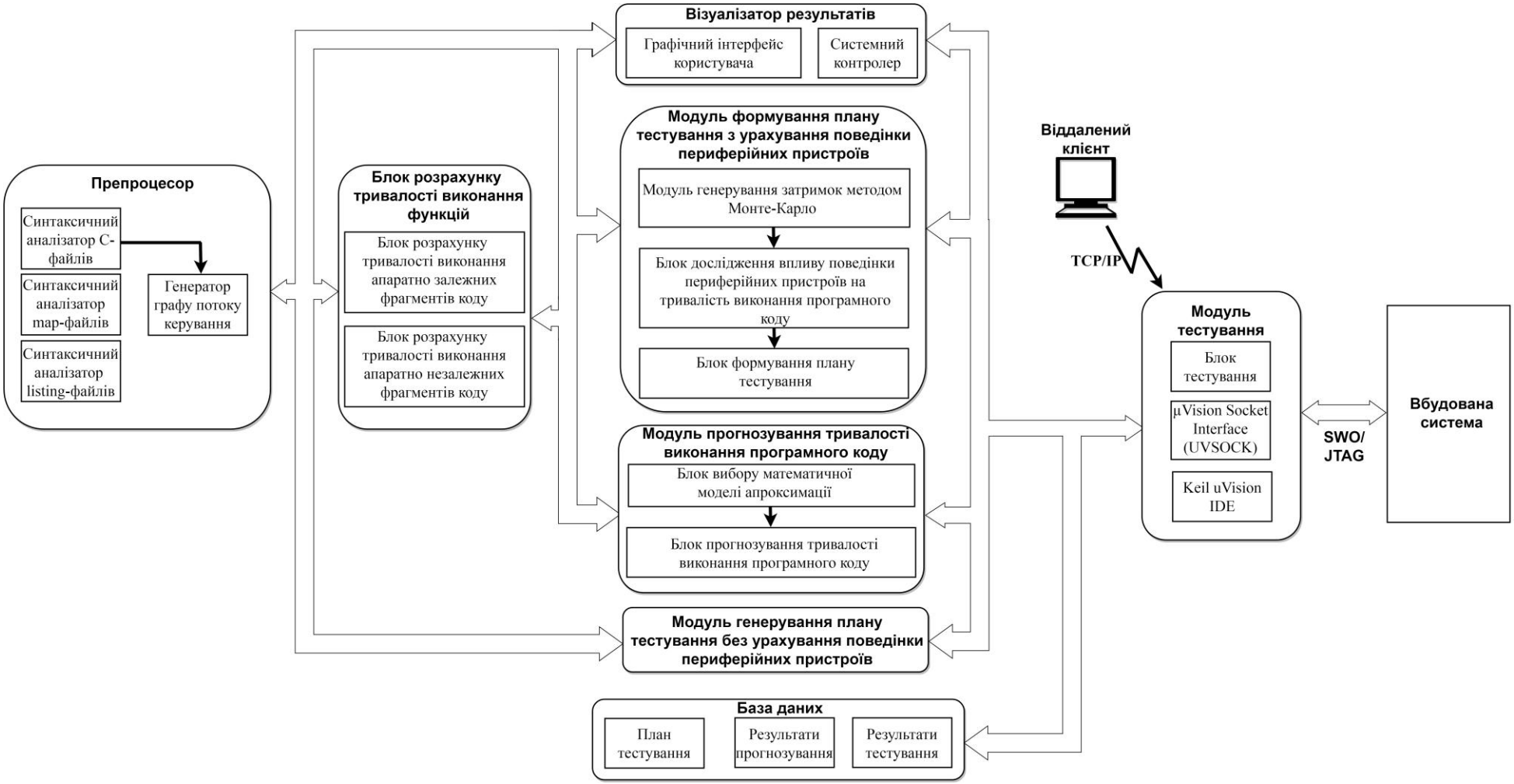


Рисунок 5.1 – Архітектура програмної системи «EXTT»

1. Препроцесор слугує для побудови графів потоку керування з метою їх подальшого аналізу. Препроцесор складається з чотирьох модулів: *синтаксичного аналізатора map-файлу, синтаксичного аналізатора С-файлів, синтаксичного аналізатора listing-файлу та генератора графу потоку керування.*

Синтаксичний аналізатор map-файлу визначає список програмних функцій, що входять до складу програмного забезпечення вбудованої системи, а відтак підлягають тестуванню. Отриманий список функцій зберігається у форматі XML для подальшої роботи.

Синтаксичний аналізатор С-файлів виконує пошук відповідної функції у файлах проекту, а також попередню обробку тексту програми. На цьому етапі з тексту програми видаляються коментарі, порожні рядки, а також проводиться модифікація тексту програми, якщо виклик функції або умову галуження записано у декілька рядків. Окрім цього, синтаксичний аналізатор визначає тип вхідних та вихідних параметрів програмної функції та оновлює XML файл. Для подальшого використання іншими модулями.

Синтаксичний аналізатор listing-файлу виконує видалення тексту програмної функції написаного мовою С, залишаючи тільки асемблерні інструкції, що будуть використовуватися для подальшого розрахунку тривалості виконання програмного коду.

Генератор графу потоку керування виконує побудову графу за текстом програми, та зберігає його у вигляді матриці інцидентності [100]. Отриманий граф потоку керування використовується для: визначення кількості гілок програми, визначення списку змінних та їх значень, що потрібні для потрапляння у відповідну гілку програми, візуалізації графу потоку керування, що буде використовуватись при графічному представленні результатів.

2. Модуль розрахунку тривалості виконання функцій є одним з основних компонентів, що складається з таких блоків: *блок розрахунку тривалості виконання апаратно-незалежних фрагментів коду*, *блок розрахунку тривалості виконання апаратно-залежних фрагментів коду.*

Блок розрахунку тривалості виконання апаратно-незалежних фрагментів коду виконує визначення загальної кількості тактів, що потрібні для виконання програми, шляхом зіставлення асемблерних інструкцій з Listing-файлу програмної функції та інформації про кількість тактів, які потрібні для її виконання тієї чи іншої асемблерної інструкції [26]. Загальна ТВПК визначається, як добуток кількості тактів, що потрібні для виконання усіх асемблерних інструкцій на час виконання одного такту.

Блок розрахунку тривалості виконання апаратно залежних фрагментів коду відповідає за визначення тривалості виконання програмного коду, що залежить від часу відгуку периферійних пристроїв. Варто відзначити, що дані про час відгуку периферійних пристроїв вносяться у базу даних вручну з документації на периферійний пристрій.

3. Модуль формування плану тестування з урахуванням поведінки периферійних пристроїв забезпечує подальшу обробку даних одержаних в ході роботи модуля розрахунку тривалості виконання програмного коду. Модуль формування плану тестування складається з трьох під модулів: *модуля генерування затримок методом Монте-Карло, модуля дослідження впливу поведінки периферійних пристроїв на ТВПК та модуля формування плану тестування.*

Модуль генерування затримок відповідає за генерування значень затримок, що являють собою можливий час відгуку периферійних пристроїв, який лежить в межах, що зазначені у документації на цей периферійний пристрій.

Модуль дослідження впливу поведінки периферійних пристроїв на ТВПК виконує розрахунок тривалості виконання програмного коду використовуючи значення згенерованих затримок, а також розраховує значення середньої тривалості виконання програмної функції, середньоквадратичного відхилення тривалості виконання програмної функції, значення вагового коефіцієнта для кожного потоку програми, а також коефіцієнта ваги функції.

Модуль формування плану тестування відповідає за створення списку функій, що підлягають тестуванню, які посортовані за спаданням значення коефіцієнта ваги функції. Варто відзначити, що згенерований план тестування зберігається у базі даних, разом із розрахованими значеннями вагових коефіцієнтів для кожної функції.

4. Модуль прогнозування тривалості виконання програмного коду виконує аналіз та прогнозування тривалості виконання програмного коду з урахуванням впливу на периферійні пристрої зовнішніх та внутрішніх чинників. До складу модуля прогнозування тривалості виконання програмного коду входять два під модулі: *модуль вибору математичної моделі апроксимації та модуль прогнозування тривалості виконання програмного коду*.

Модуль вибору математичної моделі апроксимації виконує розрахунок значення похибки апроксимації, часу відгуку периферійних пристроїв при дії на них зовнішніх чи внутрішніх чинників та вибору моделі, яка забезпечує мінімальний рівень похибки апроксимації.

Модуль прогнозування тривалості виконання програмного коду виконує розрахунок тривалості виконання програмного коду, при дії на них певних зовнішніх чи внутрішніх чинників. Отримані результати прогнозування тривалості виконання програмного коду зберігаються у базі даних для їх подальшого застосування, а саме: перевірці вимогам кліматичного виконання, визначення періоду технічного обслуговування вбудованої системи.

5. Модуль формування плану тестування без урахування поведінки периферійних пристроїв виконує формування плану тестування за допомогою сортування списку програмних функцій у випадковому порядку. Сформований план тестування зберігається у базі даних, як основний план тестування, за умови відсутності плану тестування, що враховує поведінку периферійних пристроїв.

6. Бази даних. Інформація про план тестування, результати тестування та прогнозування зберігається в базі даних. Відповідно, інформаційний вміст бази даних представляється трьома модулями: план тестування, результати

тестування, результати прогнозування. На інформаційному рівні кожний з цих модулів відповідає кільком взаємопов'язаним реляційним таблицям. З базою даних взаємодіють усі модулі програмної системи «EXTT».

Засіб «EXTT» допускає використання будь-якої реляційної бази даних, що підтримує стандартну мову SQL для маніпулювання даними.

7. Візуалізатор результатів виконує системний контроль над виконанням процесу тестування тривалості виконання програмного коду, використання потрібного програмного модуля, а також зображенню отриманих результатів тестування.

8. Модуль тестування призначений для тестування тривалості виконання програмного коду. Для реалізації тестування тривалості модуль тестування містить три модулі: *блок тестування, µVision Socket Interface API та Keil µVision IDE*.

Блок тестування виконує вимірювання тривалості виконання програмного коду у режимі відлагодження програми. Доступ до функцій відлагодження середовища розроблення програмного забезпечення Keil µVision, виконується через інтерфейс µVision Socket Interface API.

Результати тестування оновлюються у базі даних, та наводяться користувачу у вигляді звіту про тестування.

9. Віддалений клієнт слугує для забезпечення можливості тестування вбудованої системи через мережу Інтернет. Особливістю віддаленого клієнта є потреба інсталяції програмної системи «EXTT» на стороні клієнта та сервера. Вся взаємодія з вбудованою системою виконується за допомогою стандартного протоколу TCP/IP та протоколу, що входить до складу µVision Socket API.

5.2 Опис обраних технологій для програмної реалізації та користувацького інтерфейсу

З огляду на тісний взаємозв'язок програмної системи з середовищем розроблення програмного коду вбудованих систем Keil µVision, а також те, що воно працює тільки з операційною системою Windows [101], для розроблення програмного засобу «EXTT» обрано: мову програмування C# (об'єктно-

орієнтована мова програмування з безпечною системою типізації для платформи .NET [102]), технологію WPF (Windows Presentation Foundation) для реалізації графічного інтерфейсу, пакети SharpSvn (призначений для аналізу інформації з систем контролю версій ПЗ вбудованих систем), µVision Socket Interface (бібліотека, що призначена для надання доступу до функцій середовища розроблення ПЗ Keil µVision з сторонніх програмних продуктів).

Розроблений інтерфейс користувача включає в себе мінімальний набір компонентів, що робить його простим та зручним у користуванні, в той самий час, цей інтерфейс дає змогу користувачу повною мірою застосовувати весь функціонал розробленого засобу.

Інтерфейс користувача складається з основного та додаткових вікон, що репрезентують результати роботи незалежних програмних модулів. Основне вікно застосунку (рис. 5.2) містить п’ять основних клавіш, а також поле для введення шляху розміщення проекту з ПЗ вбудованої системи. Клавіша "Аналіз програмного коду" виконує синтаксичний аналіз c, map та listing файлів проекту, побудову графу потоку керування, а також підрахунок кількості гілок у функції та кількості асемблерних операцій. Окрім цього, у вікні наведено не тільки список функцій, що входить до складу проекту, а і зображення графу потоку керування під час вибору певної функції (рис. 5.3).

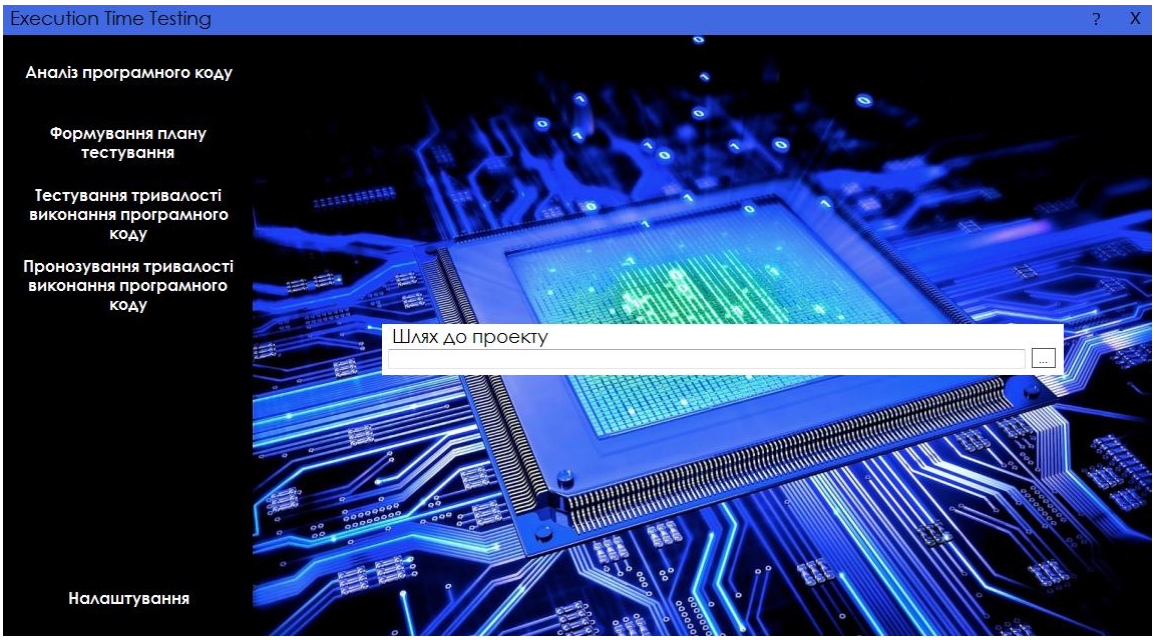


Рисунок 5.2 – Головне меню програмної системи «EXTT»

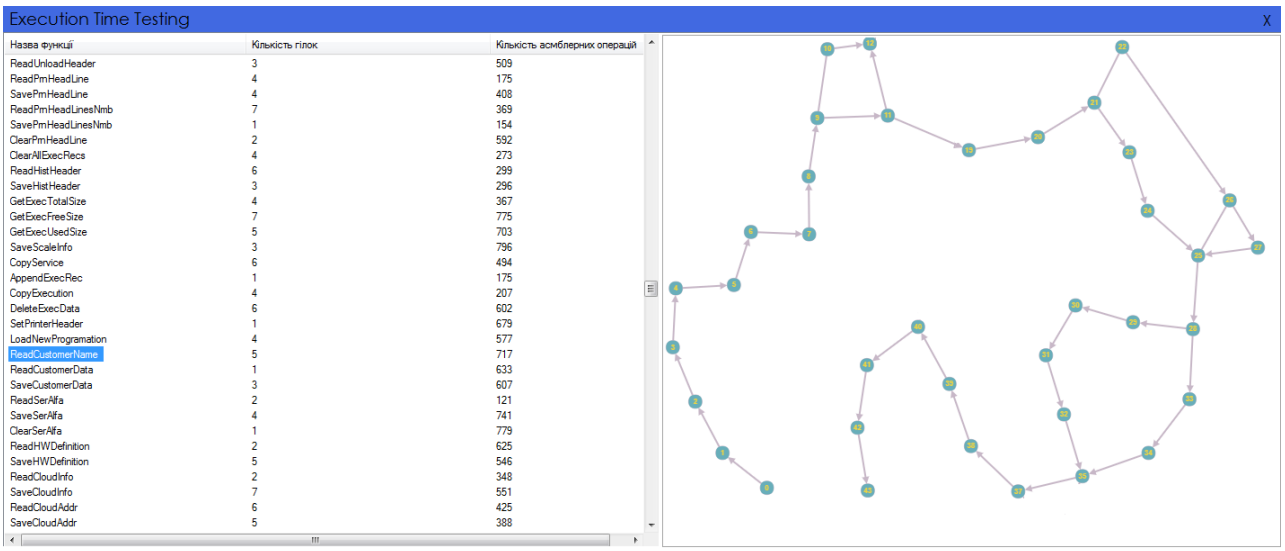


Рисунок 5.3 – Користувацький інтерфейс екрану, що відображає результати синтаксичного аналізу програм

Етап аналізу програмного коду є обов'язковим кроком, при будь-якому використанні розробленого програмного засобу, однак етапи формування плану тестування, тестування ТВПК, а також прогнозування ТПВК є незалежними та не передбачають певної послідовності застосування, що підвищує гнучкість застосування розробленої програмної системи.

Для налаштування програмної системи розроблене відповідне підменю (рис. 5.4), що містить основні налаштування, такі як: мова користувацького інтерфейсу, тип алгоритму тестування, шлях до бази даних, а також шлях звіту з результатами тестування.

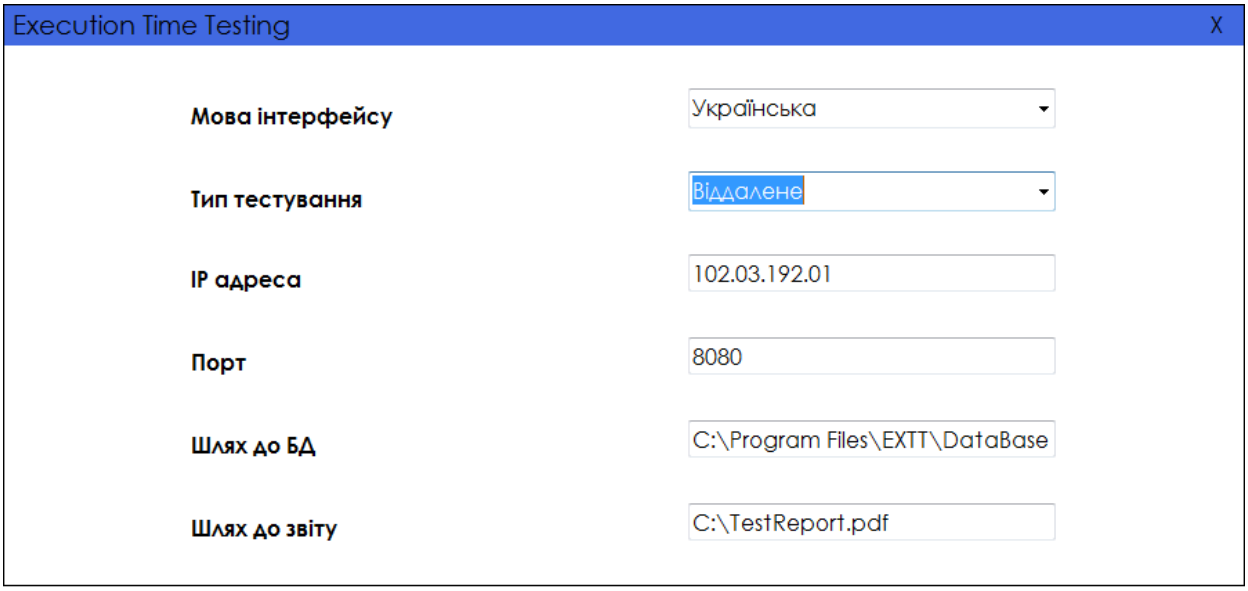


Рисунок 5.4 – Підменю налаштувань програмної системи «EXTT»

5.3 Формування плану тестування з урахуванням поведінки периферійних пристроїв

Для ілюстрації процесу створення плану тестування обрано один з продуктів компанії Dinamica Generale – Smart Control, що керує сільськогосподарськими гідравлічними системами. Для формування плану тестування, потрібно виконати певну послідовність дій, а саме:

- вказати шлях до проекту у головному меню програми;
- натиснути клавішу "Аналіз програмного коду", для здійснення синтаксичного аналізу файлів проекту;
- натиснути клавішу "Формування плану тестування" у головному меню програми;
- У підменю "Формування плану тестування" обрати алгоритм генерування плану тестування та натиснути клавішу "Створити план тестування";
- У діалогове вікно (рис. 5.5) заносимо інформацію про поведінку периферійних пристрої, а саме номінальне, максимальне та мінімальне значення часу відгуку;
- Створений план тестування, може, бути збережений, як окремий документ у форматі PDF. Для цього у вікні, що відображає план тестування (рис. 5.6) потрібно натиснути клавішу "Зберегти" та вказати шлях за яким його зберегти.

Вході виконання синтаксичного аналізу проаналізовано 143562 рядки програмного коду, що являють собою 1543 програмних функції. На підставі проведеного аналізу розраховано вагові коефіцієнти для 12 потоків програми, а також значення ваги кожної програмної функції. Визначено список з 61 функції, що характеризуються найвищим значення ваги, а відтак підлягають більш ретельному тестуванню.

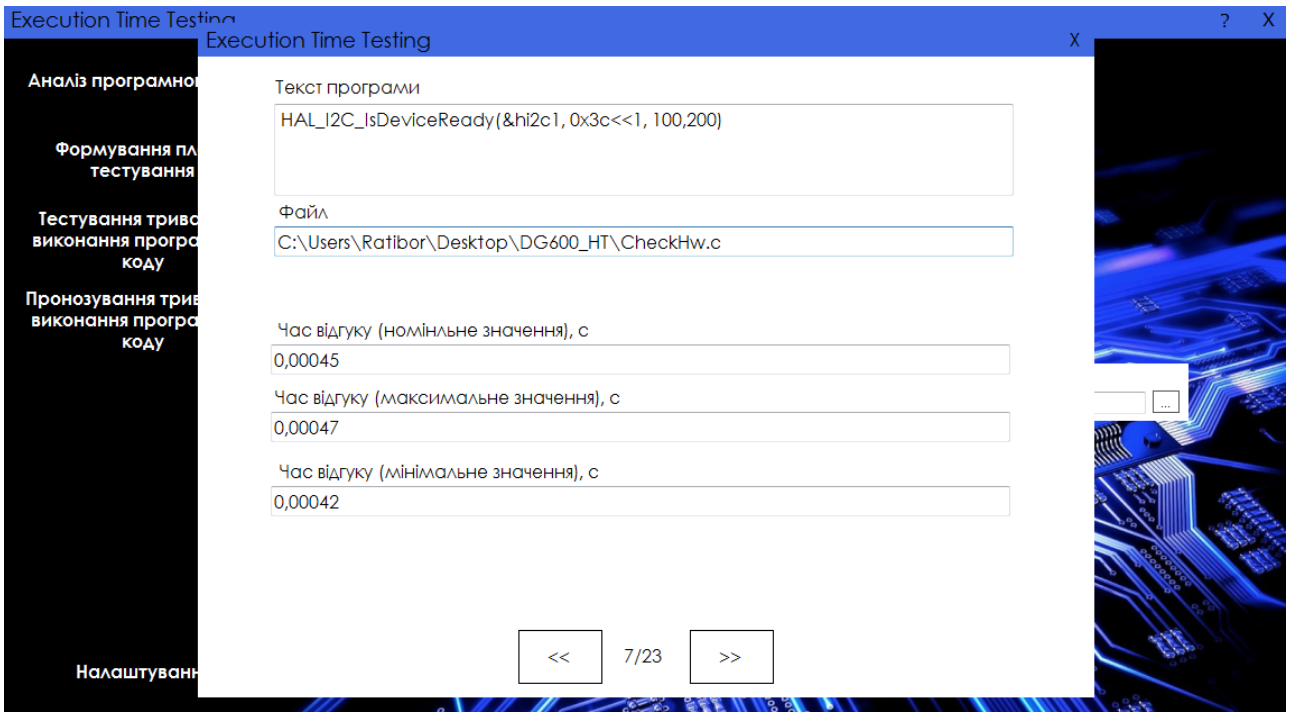


Рисунок 5.5 – Діалогове вікно програмної системи «EXTT» для введення інформації про поведінку периферійних пристроїв

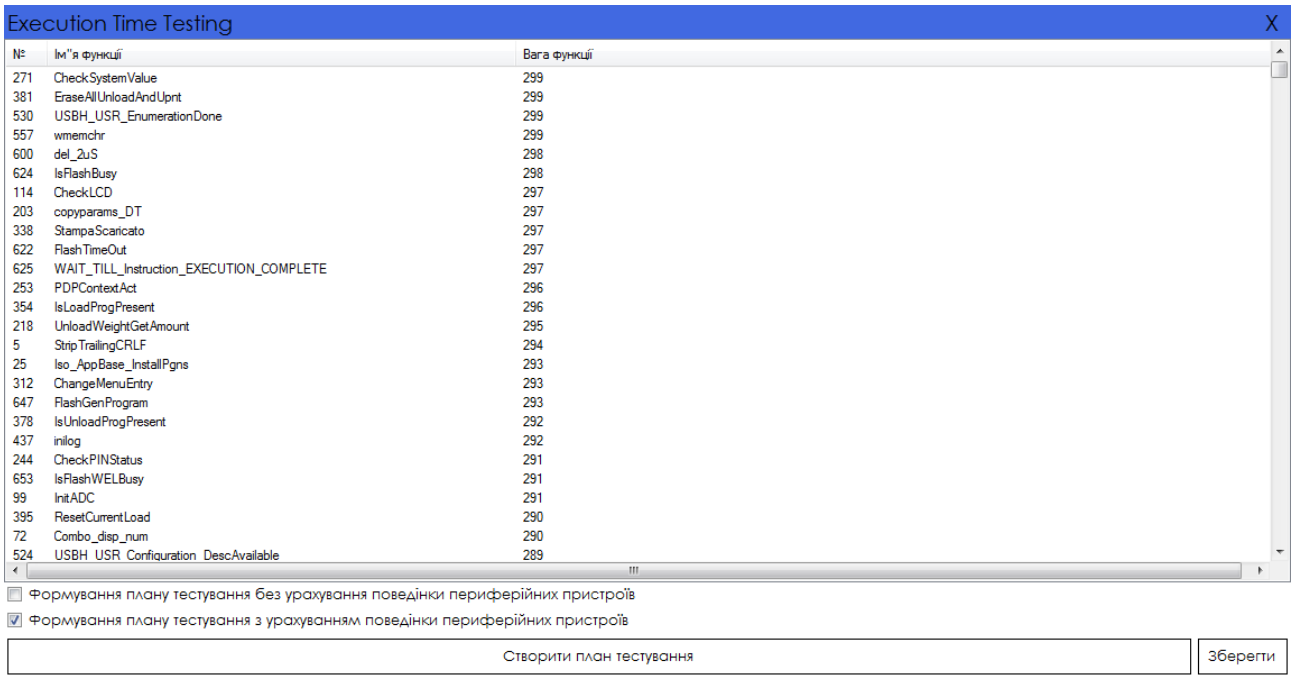


Рисунок 5.6 – Підменю формування плану тестування програмної системи «EXTT»

5.4 Автоматизоване тестування тривалості виконання програмного коду

Для тестування тривалості виконання програмного коду вбудованої системи використовуємо відповідне підменю (рис. 5.7). Для тестування вибраної програмної функції вбудованої системи потрібно виконати таку послідовність кроків:

- вибрати функцію з списку програмних функцій вбудованої системи;
- задати кількість вимірювань, що потрібно виконати;
- задати початкові значення змінним, які передаються, як параметри функції;
- натиснути клавішу "Виконати тестування"

Після завершення тестування найбільше значення тривалості виконання програмної функції буде відображено у полі – результат вимірювання.

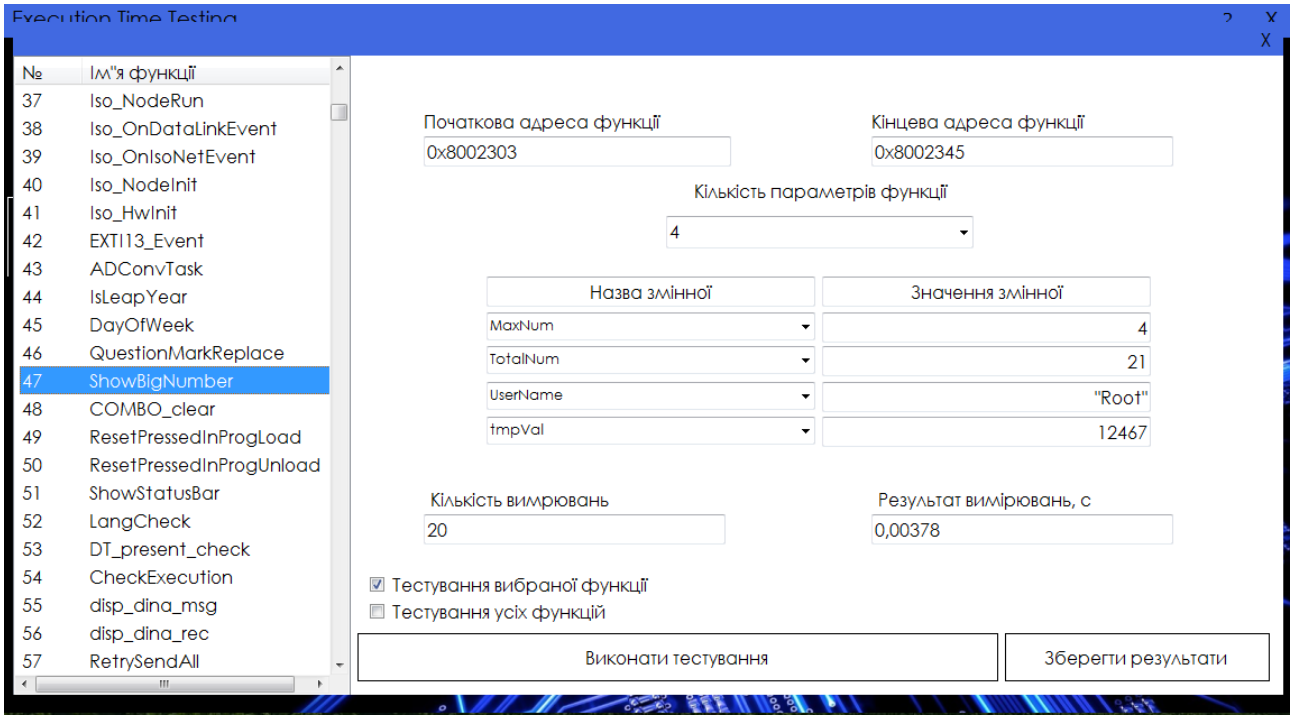


Рисунок 5.7 – Підменю тестування тривалості виконання програмної функції

Окрім цього, розроблене ПЗ дає змогу виконати автоматизоване тестування тривалості виконання для усіх програмних функцій вбудованої системи. Для цього потрібно встановити прапорець "тестування усіх функції" та натиснути клавішу "Виконати тестування". За результатами проведеного

тестування буде сформовано звіт, який міститиме результати тестувння, а також мінімальне, максимальне та середнє значення ТВПК для кожної програмної функції. Для збереження звіту про результати тестування у форматі PDF, потрібно натиснути відповідну клавішу.

5.5 Прогнозування тривалості виконання програмного коду з урахуванням зовнішніх та внутрішніх чинників

Передумовою для здійснення прогнозування тривалості виконання програмного коду є введення у базу даних інформацію про компоненти вбудованої системи, а саме назву компонента, номінальне значення часу відгуку, та дані про залежність часу відгуку від певного зовнішнього чи внутрішнього чинника (рис. 5.8).

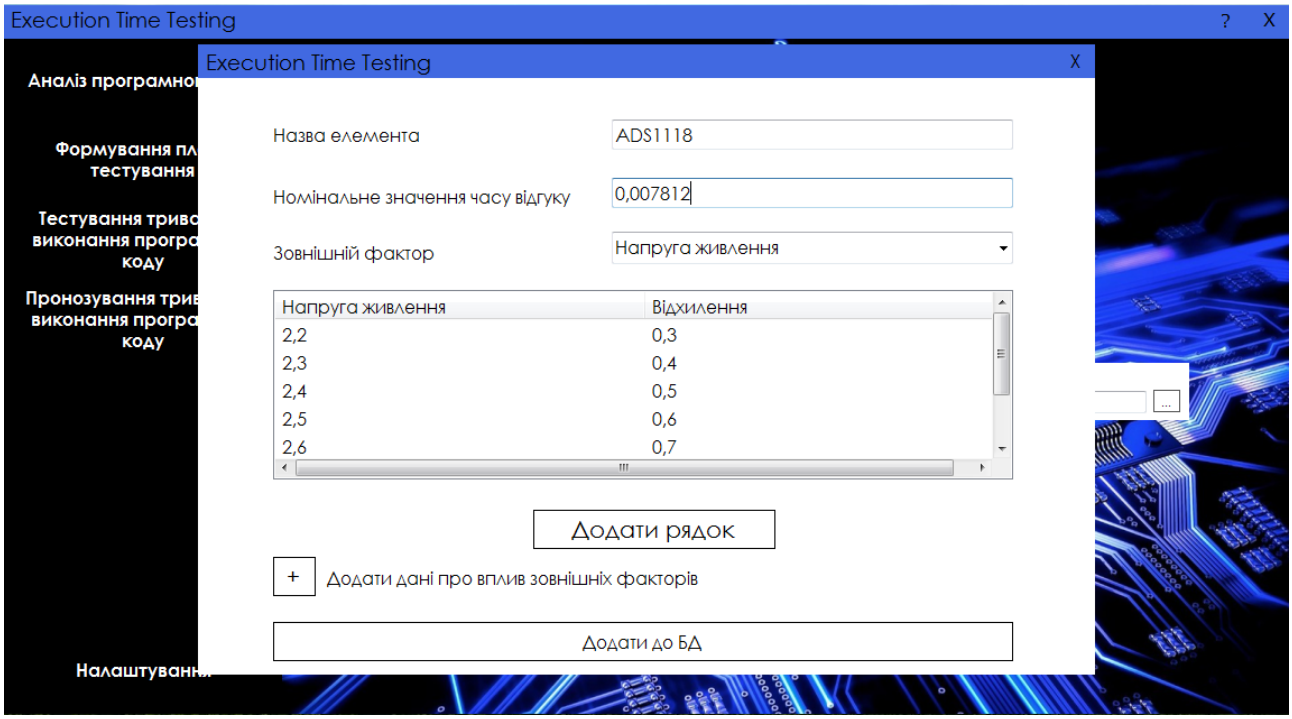


Рисунок 5.8 – Підменю введення інформації про компоненти вбудованої системи

Введені дані використовуються для розрахунку коефіцієнтів апроксимації та вибору методу апроксимації, що забезпечує найменше значення похибки апроксимації.

Після введення інформації про компоненти вбудованої системи виконується розрахунок тривалості виконання обраної програмної функції під

час дії на неї зовнішніх чи внутрішніх чинників в заданих межах. Для цього у під меню "Прогнозування тривалості виконання програмного коду" (рис 5.9), обираємо функцію із списку функцій проекту, чинник, що здійснює вплив на компоненти, метод апроксимації, а також вносимо дані про діапазон прогнозування. Проведення розрахунку та побудова залежності, тривалості виконання програмного коду при дії на нього обраного чинника, виконується після натиснення клавіші "Виконати розрахунок".

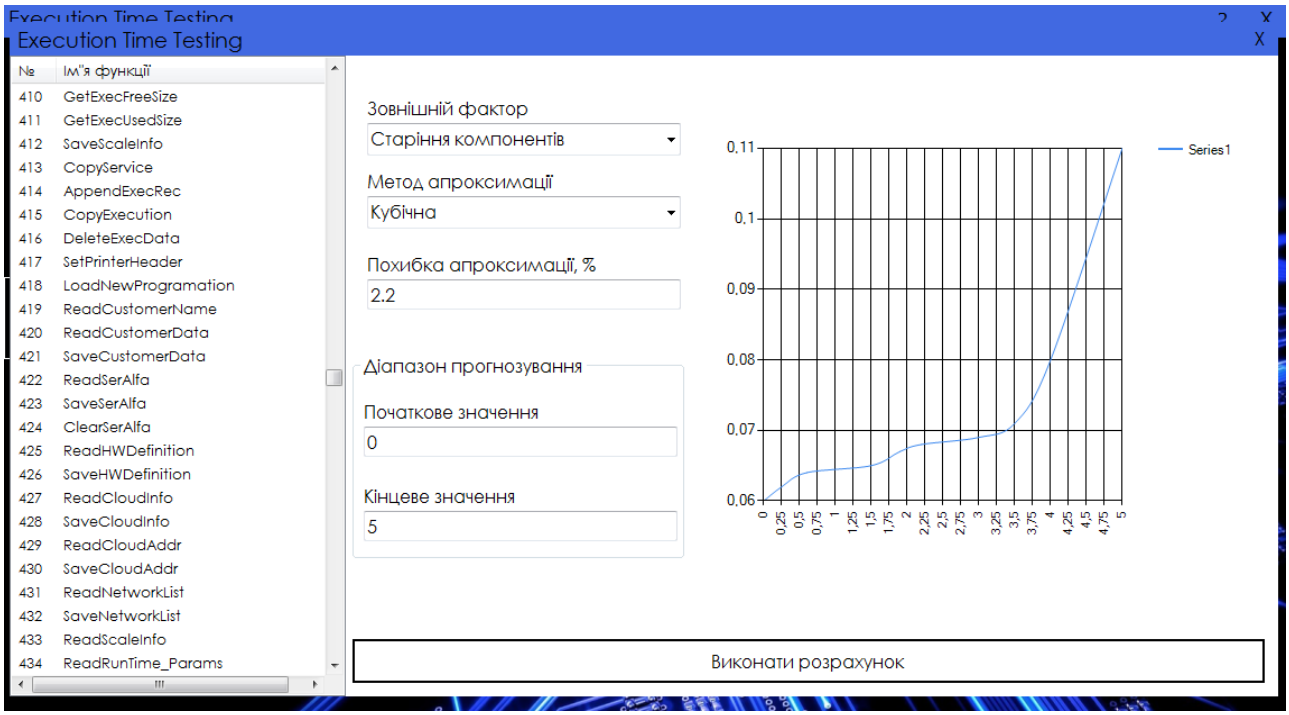


Рисунок 5.9 – Підменю прогнозування ТВПК «EXTT»

Для прогнозування ТВПК використано програмне забезпечення вбудованої системи Smart Control компанії Dinamica Generale. Проведений розрахунок показав, що тривалість виконання функції, яка відповідає за аварійну зупинку гідравлічної системи, збільшується впродовж усього періоду експлуатації системи. Отримана графічна залежність тривалості виконання програмного коду від періоду експлуатації, може, бути використана для визначення періоду експлуатації ТО вбудованої системи, або періоду роботи після якого система має бути виведена з експлуатації.

5.6 Порівняння розробленого програмного засобу та програмного засобу RapiTime

Для порівняння розробленого програмного засобу «EXTT» та «RapiTime» проведено тестування п’яти вбудованих систем Італійської компанії Dinamica Generale, а саме:

Tomato Sniper – це вбудована система, що призначена для сортування та відбору за кольором овочів та фруктів (рис 5.10). Ця система дає змогу виявляти та усувати з сортувальної лінії: метал, пластик, каміння, скло, деревину, стебла використовуючи пневматичні приводи, що працюють під тиском 4 Бар.

Рисунок 5.10 – Зовнішній вигляд вбудованої системи «Tomato Sniper» [28]

Враховуючи високу ймовірність травмування оператора вбудованої системи, її ПЗ містить потік програми, що відповідає за аварійну зупинку пневматичних клапанів у разі натискання відповідної клавіші.

Smart Control – це програмований ваговий індикатор, з можливістю керувати будь-якими сільськогосподарськими гідравлічними системами. Ця система складається з декількох пристроїв, а саме: монітор для керування системою та модуля вимірювання ваги і управління гідравлічними клапанами (рис 5.11). Ці пристрої зв’язуються між собою по радіоканалу, що призводить

до потреби введення додаткових засобів безпеки для користувача, а саме: примусова зупинка керування гідравлічною системою при виході за межі покриття сигналу від модуля управління, аварійна зупинка гідравлічної системи, діагностика критичних з точки зору безпечності модулів під час кожного запуску системи.

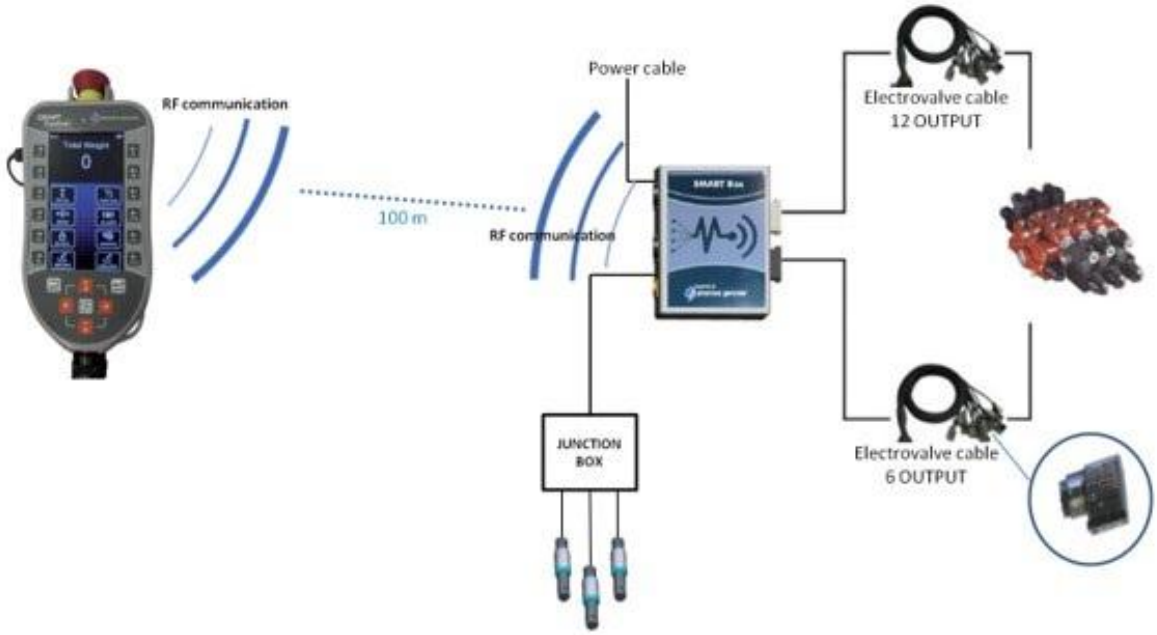


Рисунок 5.11 – Зовнішній вигляд системи «Smart Control» [28]

Pump Logger – це вбудована система, що призначена для відстеження стану промислових насосів, збереження даних про зміну стану, а також їх управління у випадку надзвичайних ситуацій (рис. 5.12). Відкриття аварійних клапанів у випадку перевищення максимально допустимого тиску чи температури всередині насосу.

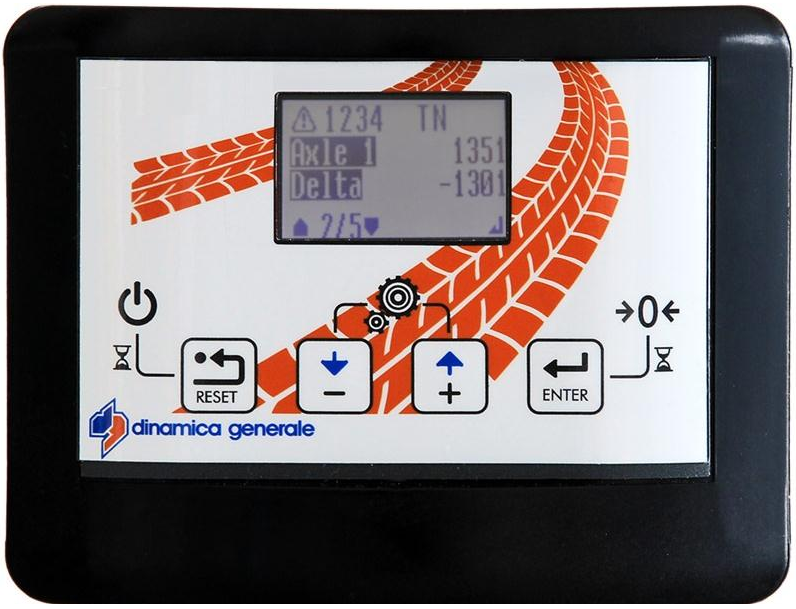


Рисунок 5.12 – Зовнішній вигляд вбудованої системи «Pump Logger» [28]

Field Scale – це багатоцільовий ваговий індикатор, що розроблений для керування процесом збору врожаю в полі, а саме: отримання інформації про поточну вагу зібраного врожаю, його хімічні параметри (вологість, рівень протеїну, крохмалю, ADF, NDF та інших), координати за якими проведено збір врожаю, а також керування процесом вивантаження зібраного врожаю (рис. 5.13).

Рисунок 5.13 – Зовнішній вигляд вбудованої системи «Field Scale» [28]

Можливість керуванням процесом вивантаження зібраного врожаю передбачає використання додаткових модулів, що здійснюють управління гідравлічними чи пневматичними клапанами. Саме тому, вбудована система Field Scale містить у собі функції, що зупиняють процес відкривання чи закривання клапанів під час виникнення аварійних ситуацій.

DG8000-IC – це багатоцільовий ваговий індикатор, який розроблений для підвищення ефективності процесу годування тварин. Ця система призначена для виконання наступних функцій: створення рецептів для годування, виконання процедури завантаження інгредієнтів, що входять до складу рецепта, проведення хімічного аналізу інгредієнтів для визначення їх хімічних параметрів, проведення змішування завантажених інгредієнтів, та вивантаження корму у встановленому місці (рис. 5.14).

Рисунок 5.14 – Зовнішній вигляд вбудованої системи «*DG8000-IC*» [28]

Усі наведені вбудовані системи призначені для застосування у різних галузях, а відтак містять у собі різні периферійні пристрої час відгуку яких впливає на ТВПК. Спільним для усіх цих системи є те, що кожна з них є системою РЧ, саме тому вона містить одну або декілька, критичних з точки

зору безпечності, функцій тривалість виконання яких не повинна перевищувати значення, що вказане у специфікації.

Порівняння розробленого програмного засобу «EXTT» та «RapiTime» проводилося за декількома ключовими критеріями: точність отриманих результатів, час витрачений на тестування, та можливість прогнозування результатів.

Результати проведеного дослідження подані у табл. 5.1, 5.2.

Таблиця 5.1 – Дослідження точності вимірювання ТПВК

Назва вбудованої системи	Виміряне значення ТВПК з використанням «EXTT», с	Виміряне значення ТПВК з використанням «RapiTime», с	Похибка вимірювання, %
Tomato Sniper (ВС 1)	0,00252	0,00245	3
Smart Control (ВС 2)	0,00743	0,00715	4
Pump Logger (ВС 3)	0,01417	0,01368	3,6
Field Scale (ВС 4)	0,01897	0,01821	4,2
DG8000-IC (ВС 5)	0,00916	0,00877	4,7

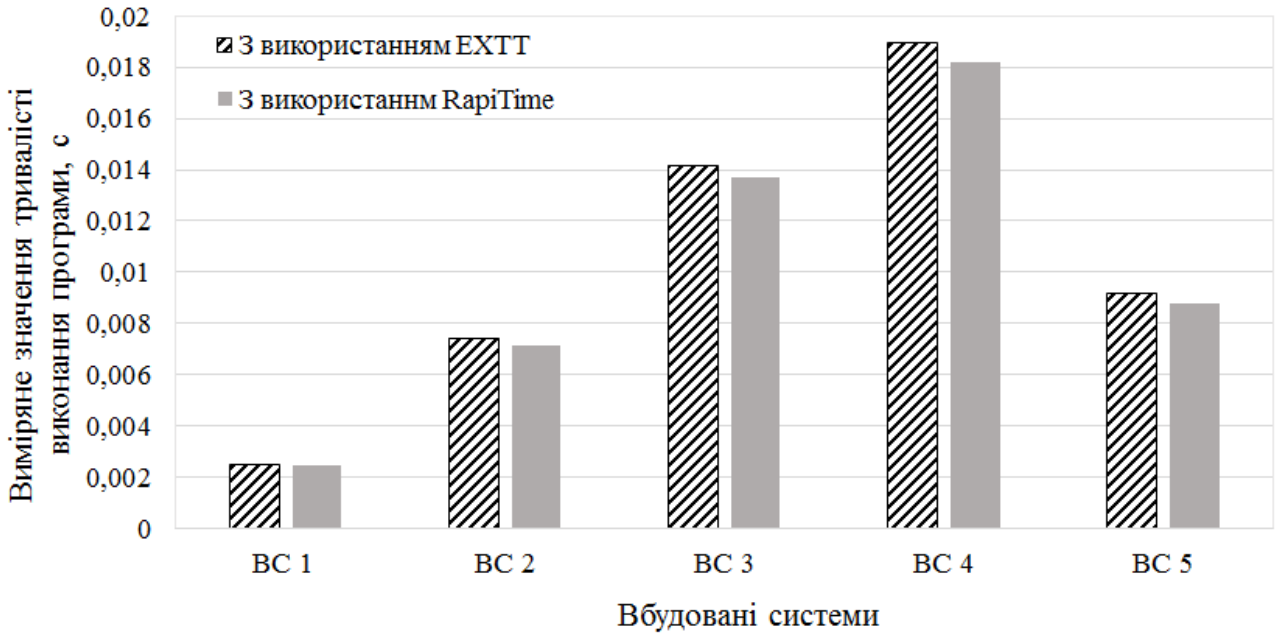


Рисунок 5.15 – Розподіл точності отриманих результатів при використанні різних програмних засобів

Як видно з рис. 5.15 використання програмного засобу «EXTT» дає змогу підвищити точність отриманих результатів для всіх вбудованих систем.

Середнє підвищення точності отриманих результатів сягає 3,8 %, що дає змогу підвищити рівень адекватності отриманих результатів. Крім цього, підвищення точності вимірювання приводить до підвищення рівня безпечності вбудованої системи.

Таблиця 5.2 – Дослідження витраченого на тестування часу при використанні різних програмних засобів

Назва вбудованої системи	Тривалість тестування при використанні «EXTT», хв	Тривалість тестування при використанні «RapiTime», хв	Зменшення витрат часу на тестування, %
Tomato Sniper (ВС 1)	43	49	14
Smart Control (ВС 2)	65	73	13
Pump Logger (ВС 3)	71	81	15
Field Scale (ВС 4)	78	86	11
DG8000-IC (ВС 5)	58	66	15

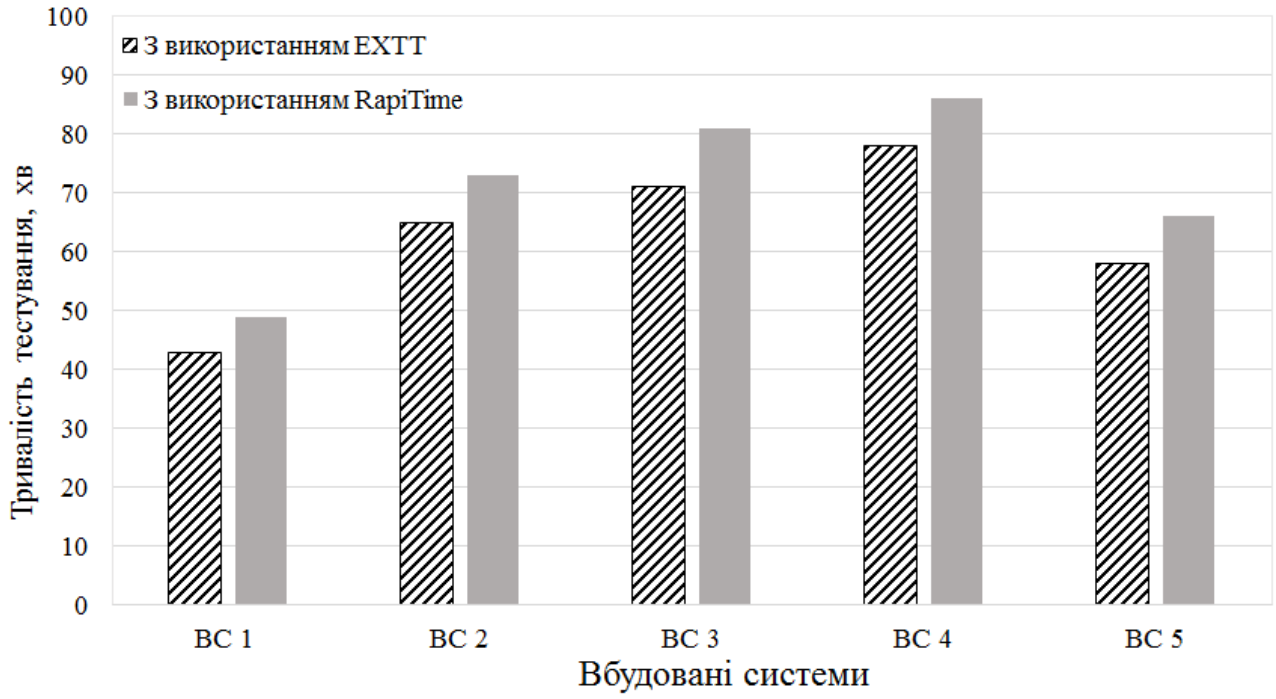


Рисунок 5.16 – Розподіл тривалості тестування при використанні різних програмних засобів

З рис. 5.16 видно, що застосування програмного засобу «EXTT» дає змогу зменшити час витрачений на тестування тривалості виконання програмного коду завдяки застосуванню методу генерування плану тестування, який враховує поведінку периферійних пристроїв. Подальше зниження витрат часу на тестування можливе за рахунок врахування ймовірностей виконання гілок програми під час створення плану тестування. Окрім цього, розроблений програмний засіб «EXTT» на відміну від «RapiTime» дає змогу врахувати вплив зовнішніх і внутрішніх чинників, що діють на вбудовану систему та призводять до зміни ТВПК. Саме це, значно підвищує ефективність застосування «EXTT», а також розширює напрямки його застосування.

5.7 Висновки до розділу

1. Розроблено програмний засіб «EXTT», що призначений для автоматизованого тестування тривалості виконання програмного коду вбудованих систем, а також прогнозування тривалості виконання програмного коду.

2. Розроблене ПЗ надає: засоби синтаксичного аналізу файлів проекту вбудованої системи, що розроблені у середовищі програмування Keil µVision, засоби формування плану тестування програмних функцій вбудованої системи з урахуванням поведінки периферійних пристроїв, засоби аналізу та прогнозування тривалості виконання програмного коду вбудованих систем з урахуванням впливу зовнішніх та внутрішніх чинників на вбудовану систему, засоби динамічного тестування тривалості виконання програмних функцій вбудованих систем.

3. Для реалізації програмної системи, обрано мову програмування C#, набір бібліотек та системних компонентів .Net Framework 4.0, та бібліотеки Math.Net, NGraphics, SkiaSharp та uVisison Socket API.

4. Використання розроблених методів, алгоритмів для тестування та прогнозування тривалості виконання програмного коду вбудованих систем жорсткого РЧ підтвердили їх ефективність, а саме експериментально доведено

збільшення точності отриманих результатів на 3–5 %, водночас зменшено витрати часу на 15%.

ОСНОВНІ РЕЗУЛЬТАТИ ТА ВИСНОВКИ

У дисертаційній роботі розв’язано актуальне наукове завдання – удосконалення наявних методів тестування тривалості виконання програмного коду вбудованих систем реального часу і створення відповідного програмного засобу для автоматизації процесу їх тестування. Отримано такі основні наукові та практичні результати:

1. На підставі результатів аналізу проблеми тестування програмного забезпечення мікроконтролерних вбудованих систем реального часу обґрунтовано необхідність удосконалення наявних методів та засобів тестування тривалості виконання програмного коду.

2. Удосконалено метод статичного аналізу тривалості виконання програмного коду для мікроконтролерних вбудованих систем, а саме запропоновано метод аналізу втрат швидкості надсилання даних через зовнішні інтерфейси зв’язку та між внутрішніми модулями мікроконтролера. Це стало можливим завдяки статичному аналізу низькорівневих функцій, які виконують налаштування мікроконтролера під час його ввімкнення.

3. Отримав подальший розвиток метод динамічного тестування тривалості виконання програмного коду, який на відміну від наявних, забезпечує повний контроль над вбудованою системою без внесення змін у її програмне чи апаратне забезпечення, що дає можливість підвищити точність отриманих результатів на 3-5%.

4. Отримав подальший розвиток метод формування плану тестування програмних функцій вбудованої системи, який, на відміну від наявних, враховує поведінку периферійних пристроїв, що дає можливість більш ефективно тестувати вбудовані системи. Експериментально встановлено, що використання плану тестування, який враховує поведінку периферійних пристроїв, забезпечує зменшення витрат часу на тестування на 11-15%.

5. Розроблено метод прогнозування тривалості виконання програмного коду, який на відміну від наявних, враховує вплив зовнішніх та

внутрішніх чинників на апаратне забезпечення вбудованих систем, та тривалість виконання їхнього програмного коду.

6. Розроблено програмний засіб для тестування тривалості виконання програмного коду вбудованих систем реального часу, що дає змогу автоматизувати процес тестування вбудованих систем, а також здійснювати розрахунок періодичності технічного обслуговування вбудованих систем, що забезпечить заданий рівень надійності та безпечності їх експлуатації.

7. Результати дисертаційної роботи впроваджено в процес тестування вбудованих систем у компанії Dinamica Generale S.p.A, ПП НВПП “Спаринг-Віст Центр” та в навчальний процес Національного університету «Львівська політехніка».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Worst Case Timing Analysis Technique for Multiple-Issue Machines / S.Lim, J. Han, J. Kim, S. Min. // 19th IEEE Real-Time Systems Symposium (RTSS'98). – 1998.
2. About the Serial Wire Output [Електронний ресурс] // ARM Information Center – Режим доступу до ресурсу: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Chdhgebe.html.
3. Adding Instruction Cache Effects to Schedulability Analysis of Preemptive Real-Time System / [J. Busquets-Mataix, J. Serrano, R. Ors та ін.]. // 2nd IEEE Real-Time Technology and Applications Symposium (RTAS'96). – 1996. – С. 204–212.
4. ADS1118 Ultrasmall, Low-Power, SPI-Compatible, 16-Bit Analog-to-Digital Converter with Internal Reference and Temperature Sensor datasheet (Rev. E) [Електронний ресурс] // Texas Instruments – Режим доступу до ресурсу: http://www.ti.com/product/ADS1118.
5. Aljifri H. Tighten the computation of worst-case execution-time by detecting feasible paths / H. Aljifri, A. Pons, M. Tapia. // Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference. – 2000. – С. 23–34.
6. An Accurate Worst Case Timing Analysis Technique for RISC Processors/ [S. Lim, Y. Bae, G. Jang та ін.]. // 15th Real-Time Systems Symposium. – 1994. – С. 49 – 51.
7. An accurate worst-case timing analysis for RISC processors / [S. Lim, Y. Bae, C. Jang та ін.]. // IEEE Transactions on Software Engineering. – 1995. – №21. – С. 593–604.
8. ARM CoreSight [Електронний ресурс] // ARM Information Center – Режим доступу до ресурсу:

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0528b/index.html&_ga=2.151482042.280633297.1539027660-658444056.1537282439.

9. Atanassov P. Impact of DRAM Refresh on the Execution Time of Real-Time Tasks / P. Atanassov, P. Puschner. // Pacific Rim International Symposium on Dependable Computing (PRDC'2001). – 2001.
10. Bundala D. On Systematic Testing for Execution-Time Analysis / D. Bundala, S. A. Seshia // In IEEE Real-Time Embedded Systems Workshop, held in conjunction with RTSS2015. – 2015.
11. Atanassov P. Using Real Hardware to Create an Accurate Timing Model for Execution-Time Analysis / P. Atanassov, R. Kirner, P. Puschner. // In IEEE Real-Time Embedded Systems Workshop, held in conjunction with RTSS2001. – 2001.
12. Automatic timing model generation by CFG partitioning and model checking / I.Wenzel, B. Rieder, R. Kirner, P. Puschner. // Design, Automation and Test in European Conference and Exhibition (DATE'05). – 2005. – С. 606–611.
13. Bernat G. pWCET: a tool for probabilistic worst-case execution time analysis / G. Bernat, A. Colin, S. Petters. // ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'03). – 2003.
14. Bounding pipeline and instruction cache performance / [C. Healy, F. Arnold, D. Whalley та ін.]. // IEEE Transactions on Computers (Volume: 48 , Issue: 1). – 1999. – №1. – С. 53 – 70.
15. Optimizing instruction cache performance for operating system intensive workloads / [J. Torrellas, Chun Xia, R. Daigle]. // 1st IEEE Symposium on High Performance Computer Architecture. – 1995. – С. 17–25.
16. Bounding Worst-Case Instruction Cache Performance / R.Arnold, F. Mueller, D. Whalley, M. Harmon. // Proceedings Real-Time Systems Symposium. – 1994. – С. 172–181.

17. Breakpoints, watchpoints, and exceptions [Електронний ресурс] // Free Software Foundation – Режим доступу до ресурсу: http://web.mit.edu/gnu/doc/html/gdb_7.html.
18. CAST: Automating Software Tests for Embedded Systems / M.Wahler, E. Ferranti, R. Steiger, R. Jain. // IEEE Fifth International Conference on Software Testing, Verification and Validation. – 2012. – С. 123–133.
19. Chopey R. The model of software execution time remote testing [Text]. / R. Chopey, B. Knysh, D. Fedasyuk // Computer Science and Engineering 2017 of the IXth International Conference of Young Scientists CSE'2017, November 22 – 24. – Lviv, 2017. – P. 398 – 402.
20. Chopey, R. A Model For Estimating Firmware Execution Time Based On Peripherals' Time Behavior [Текст]. / R. Chopey, D. Fedasyuk, T. Marusenkova // Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering of the XIVth IEEE International Conference TCSET'2018, February 20 – 24. – Lviv, 2018. – С. 1179 – 1183.
21. Colin A. A Modular and Retargetable Framework for Tree-Based WCET Analysis / A. Colin, I. Puaut. // 13th Euromicro Conference of Real-Time Systems, (ECRTS'01). – 2001.
22. Colin A. Scope-tree: a program representation for symbolic worst-case execution time analysis / A. Colin, G. Bernat. // 14th Euromicro Conference of Real-Time Systems, (ECRTS'02). – 2002. – С. 50–59.
23. Colin A. Worst Case Execution Time Analysis for a Processor with Branch Prediction / A. Colin, I. Puaut. // Journal of Real-Time Systems. – 2000. – №18. – С. 249–274.
24. Colin A. Worst-Case Execution Time Analysis for the RTEMS Real-Time Operating System / A. Colin, I. Puaut. // 13th Euromicro Conference of Real-Time Systems, (ECRTS'01). – 2001.
25. Control Flow Graph [Електронний ресурс] // GCC, the GNU Compiler Collection – Режим доступу до ресурсу: http://gcc.gnu.org/onlinedocs/gccint/Control-Flow.html.

26. Cortex-M3 instructions [Електронний ресурс] // ARM Information Center – Режим доступу до ресурсу: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337h/CHDDIGAC.html&_ga=2.242217546.99566189.1533813352-1242376419.1533813344.
27. Cousot P. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints / P. Cousot,, R. Cousot:. // Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA. – 1977. – С. 238–252.
28. DINAMICA GENERALE SPA [Електронний ресурс] – Режим доступу до ресурсу: https://www.dinamicagenerale.com.
29. Effect of Aging on Response Time of Nuclear Plant Pressure Sensors – Washington DC: U.S. Nuclear Regulatory Commission, 1989.
30. Embedded Software: Know It All / [J. Labrosse, J. Ganssle, R. Oshana та ін.]. – Oxford: Elsevier, 2008. – 793 с. ISBN-13: 978-0750685832
31. Embedded System Market: Global Industry Analysis and Opportunity Assessment 2016-2026 [Електронний ресурс] // Future Market Insights. – 2018. – Режим доступу до ресурсу: https://www.futuremarketinsights.com/reports/embedded-system-market.
32. Embedded Systems Market by Hardware, Software, Application, and Geography - Global Forecast to 2023 [Електронний ресурс] // Research and Markets. – 2017. – Режим доступу до ресурсу: https://www.researchandmarkets.com/research/7fs5xg/embedded_systems.
33. Engblom J. Pipeline Timing Analysis Using a Trace-Driven Simulator / J. Engblom, A. Ermedahl. // 6th International Conference on Real-Time Computing Systems and Applications (RTCSA’99). – 1999.
34. Engblom J. Structured Testing of Worst-Case Execution Time Analysis Tools / J. Engblom, F. Stappert, A. Ermedahl. // 21st Real-Time System Symposium (RTSS/WIP’00). – 2000. – С. 154–163.

35. Ermedahl A. A Modular Tool Architecture for Worst-Case Execution Time Analysis : дис. канд. техн. наук / Ermedahl Andreas – Uppsala University, Sweden, 2003.
36. Exploiting Branch Constraints without Exhaustive Path Enumeration / T.Chen, A. Roychoudhury, T. Mitra, V. Suhendra. // 5th International Workshop on Worst-Case Execution Time Analysis (WCET'05). – 2005. – С. 40–43.
37. Fan X. Real-Time Embedded Systems, Design Principles and Engineering Practices / Xiaocong Fan. – Oxford: Elsevier, 2015. – 686 с.
38. Fedasyuk D. A method of predicting the maintenance period of embedded systems for preventing breach of their time requirements / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Computing, Ukraine. 2018. – Vol. 17. – Issue 2, pp 94 – 101.
39. Fedasyuk D. A Model for Estimating Firmware Execution Time Taking Into Account Peripheral Behavior / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Intelligent Systems and Applications, Hong Kong. – 2018. – Vol. 10. – No. 6. С. 22 – 29.
40. Fedasyuk D. Determining the Influence of the External Factors on the Firmware Response Time / D. Fedasyuk, V. Gavrysh, T. Marusenkova, R. Chopey // Central European Researchers Journal, Slovakia. – 2018. – Vol. 4. – Issue 1, pp. 1 – 8.
41. Ferdinand C. Applying Compiler Techniques to Cache Behavior Prediction / C. Ferdinand, F. Martin, R. Wilhelm. // ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS'97). – 1997.
42. Ganssle J. Embedded Systems Dictionary / Ganssle. – San Francisco: CMP Books, 2003. – 305 с.
43. Global Microcontroller Embedded Systems Market Research Report-Forecast to 2027 [Електронний ресурс] // Market Research Future. – 2018. – Режим доступу до ресурсу:

https://www.marketresearchfuture.com/reports/microcontroller-embedded-systems-market-911.

44. Gustafsson J. Towards a flow analysis for embedded system C programs / J. Gustafsson, A. Ermedahl, B. Lisper. // 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005). – 2005. – С. 84–95.
45. Healy C. Tighter timing predictions by automatic detection and exploitation of value-dependent constraints / C. Healy, D. Whaley. // Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium. – 1999. – С. 79–92.
46. Henness J. Computer Architecture, A Quantitative Approach / J. Henness, D. Patterson., 2017. – 936 с.
47. Holsti N. Bound-T time and stack analyser [Електронний ресурс] / Niklas Holsti // Tidorum Ltd – Режим доступу до ресурсу: http://www.bound-t.com/.
48. Holsti N. Worst-case execution-time analysis for digital signal processors / N. Holsti, T. Langbacka, S. Saarinen. // EUSIPCO 2000 Conference (X European Signal Processing Conference). – 2000.
49. IEEE Std. 1149.1 - Standard Test Access Port and Boundary-Scan Architecture [Електронний ресурс] // IEEE Org. – Режим доступу до ресурсу: http://grouper.ieee.org/groups/1149/1/.
50. Kim S. Efficient worst case timing analysis of data caching / S. Kim, S. Min, R. Ha. // Proceedings Real-Time Technology and Applications. – 1996. – С. 230–240.
51. Kirner R. The WCET Analysis Tool CalcWcet167 / Kirner. // Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies : 5th International Symposium. – 2012. – С. 158–172.
52. Li Y. Cache modeling for real-time software: beyond direct mapped instruction cache / Y. Li, S. Malik, A. Wolfe. // 17th IEEE Real-Time Systems Symposium. – 1996. – С. 254–263.

53. Li Y. Performance analysis of embedded software using implicit path enumeration / Y. Li, S. Malik. // 32-nd Design Automation Conference. – 1995. – С. 456–461.

54. Lindgren M. Measurement and simulation based techniques for real-time systems analysis / Lindgren. // Uppsala University Printers. – 2002.

55. Lindgren M. Using Measurements to derive the Worst-case Execution Time / M. Lindgren, H. Hansson, H. Thane. // 7th International Conference on Real-Time Computing Systems and Applications (RTCSA’00). – 2000. – С. 15–22.

56. Low-level Analysis of a Portable Java Byte Code WCET Analysis Framework / I.Bate, G. Bernat, G. Murphy, P. Puschner. // 7th International Conference on Real-Time Computing Systems and Applications (RTCSA’00). – 2000. – С. 39–48.

57. Lundqvist T. An integrated path and timing analysis method based on cycle-level symbolic execution / T. Lundqvist, P. Stenstrom. // Journal of Real-Time Systems. – 2000.

58. Lundqvist T. Timing anomalies in dynamically scheduled microprocessors / T. Lundqvist, P. Stenstrom // Technical Report 99-5 / T. Lundqvist, P. Stenstrom. – Chalmers: Chalmers University of Technology, 1999.

59. Michael B. Programming Embedded Systems, 2nd Edition With C and GNU Development Tool / B. Michael, M. Anthony. – Wilmington, U.S.A.: O'Reilly Media, 2009. – 336 с.

60. Ottosson G. Worst-Case Execution Time Analysis for Modern Hardware Architectures / G. Ottosson, M. Sjodin. // ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS’97). – 1997. – С. 249–274.

61. Park C. Experiments with a program timing tool based on a source-level timing schema / C. Park, A. Shaw. // 11th IEEE Real-Time Systems Symposium (RTSS’90). – 1990. – С. 72–81.

62. Petters S. Bounding the Execution Time of Real-Time Tasks on Modern Processors / Petters. // 7th International Conference on Real-Time Computing Systems and Applications (RTCSA'00). – 2000.
63. Petters S. Making Worst-Case Execution Time Analysis for Hard Real-Time Tasks on State of the Art Processors Feasible / S. Petters, G. Farber. // 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99). – 1999.
64. Postel J. INTERNET CONTROL MESSAGE PROTOCOL [Електронний ресурс] / Postel // Network Working Group. – 1981. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc792.
65. Puaut I. Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems / I. Puaut, D. Decotigny. // 23rd IEEE International Real-Time Systems Symposium (RTSS'02). – 2002.
66. Puschner P. Computing maximum task execution times – a graph-based approach / P. Puschner, A. Schedl. // Journal of Real-Time Systems. – 1997. – №13. – С. 67–91.
67. Reliable and Precise WCET Determination for a Real-Life Processor / [C. Ferdinand, R. Heckmann, M. Langenbach та ін.]. // 1st International Workshop on Embedded Systems, (EMSOFT2000). – 2001.
68. Ringler T. Static Worst-Case Execution Time Analysis of Synchronous Programs / Thomas Ringler. // International Conference on Reliable Software Technologies. – 2000. – С. 56–68.
69. Saswat A. Demand-Driven Compositional Symbolic Execution / A. Saswat, P. Godefroid, N. Tillmann., 2008. – 381 с. – (Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science).
70. Schneider J. Cache and Pipeline Sensitive Fixed Priority Scheduling for Preemptive Real-Time Systems / Schneider. // 21th IEEE Real-Time Systems Symposium (RTSS'00). – 2000. – С. 195–204.
71. Schneider J. Pipeline Behaviour Prediction for Superscalar Processors by Abstract Interpretation / J. Schneider, C. Ferdinand. // SIGPLAN Workshop

on Languages, Compilers and Tools for Embedded Systems (LCTES’99). – 1999.

72. Software verification solutions for critical aerospace & automotive systems [Електронний ресурс] // Rapita Systems Ltd – Режим доступу до ресурсу: https://www.rapitasystems.com/.
73. Stankovic J. Misconceptions about real-time computing: a serious problem for next-generation systems / Stankovic. // Computer. – 1988. – С. 10–19. DOI: 10.1109/2.7053.
74. Stappert F. Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs / F. Stappert, P. Altenbernd. // Journal of Systems Architecture. – 2000. – №46. – С. 339–355.
75. Stappert F. Efficient longest executable path search for programs with complex flows and pipeline effects / F. Stappert, A. Ermedahl, J. Engblom. // 4th International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, (CASES’01). – 2001.
76. Stappert F. Predicting pipelining and caching behaviour of hard real-time programs / Stappert. // Proceedings Ninth Euromicro Workshop on Real Time Systems. – 1997.
77. Stewart D. Measuring execution time and real-time performance / David Stewart. // Embedded Systems Conference, (ESCSF 2004). – 2004.
78. Supporting timing analysis by automatic bounding of loop iterations / C.Healy, M. Sjodin, V. Rustagi, D. Whalley. // REAL-TIME SYSTEMS. – 2000. – С. 129–156.
79. Tavares A. A machine independent WCET predictor for microcontrollers and DSPs / A. Tavares, C. Couto. // IEEE International Symposium on Industrial Electronics Proceedings (ISIE'2001). – 2001.
80. Test Plan [Електронний ресурс] // Software Testing Fundamentals – Режим доступу до ресурсу: http://softwaretestingfundamentals.com/test-plan/.

81. The influence of processor architecture on the design and the results of WCET tools / R.Heckmann, M. Langenbach, S. Thesing, R. Wilhelm. // Proceedings of the IEEE (Volume: 91, Issue: 7). – 2003. – С. 1038 – 1054.
82. Theiling H. Combining Abstract Interpretation and ILP for Microarchitecture Modelling and Program Path Analysis / H. Theiling, C. Ferdinand. // 19th IEEE Real-Time Systems Symposium (RTSS'98). – 1998.
83. Thesing S. Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models : дис. канд. техн. наук / Thesing S. – Saarland University, 2004.
84. Timing analysis for data caches and set-associative caches / [R. White, F. Mueller, C. Healy та ін.]. // Proceedings Third IEEE Real-Time Technology and Applications Symposium (RTAS'97). – 1997. – С. 192–202.
85. Unique tools and services for the development, analysis, and certification of safety-critical software [Електронний ресурс] // AbsInt – Режим доступу до ресурсу: https://www.absint.com/.
86. Using the μVision Socket Interface [Електронний ресурс] // ARM Keil – Режим доступу до ресурсу: http://www.keil.com/appnotes/docs/apnt_198.asp.
87. Wolf F. Execution Cost Interval Refinement in Static Software Analysis / F. Wolf, R. Ernst. // Special Issue on Modern Methods and Tools in Digital System Design. – 2001. – №47. – С. 339–356.
88. Wolf F. Timing and power measurement in static software analysis / F. Wolf, J. Kruse, R. Ernst. // Microelectronics Journal, Special Issue on Design, Modeling and Simulation in Microelectronics and MEMS. – 2002. – №6. – С. 91–100.
89. York R. Embedded segment market update [Електронний ресурс] / Richard York // ARM Information Center. – 2015. – Режим доступу до ресурсу: https://www.arm.com/zh/files/event/1_2015_ARM_Embedded_Seminar_Richard_York.pdf.

90. Zhang Y. Evaluation of methods for dynamic time analysis for CC systems AB : дис. канд. техн. наук / Zhang Yina – Malardalen University, Vasteras, Sweden, 2005.
91. Zurawski R. Embedded Systems Handbook 2-Volume Set / Richard Zurawski. – Boca Raton, USA: CRC Press, 2009. – 592 с.
92. Вержбицкий В. Численные методы (математический анализ и обыкновенные дифференциальные уравнения): учеб. пособие для вузов / В. Вержбицкий., 2001. – 382 с.
93. Cortex-M3 instructions [Електронний ресурс] // ARM Information Center – Режим доступу до ресурсу: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337h/CHDDIGAC.html&_ga=2.242217546.99566189.1533813352-1242376419.1533813344
94. ГОСТ 15150-69. Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды [Текст] – Москва: Изд-во стандартов, 1971. – 57 С.
95. Федасюк Д. В. Алгоритм побудови графу потоку керування за текстом програми мовою С [Текст] / Д. В. Федасюк, Р. С. Чопей // Науковий журнал Радіоелектроніка, інформатика, управління. – Запоріжжя: Запорізький національний технічний університет, 2018. № 2. – С. 154 – 161.
96. Філософський енциклопедичний словник / [В. Шинкарук, Є. Бистрицький, М. Булатов та ін.]. – Київ: Абрис, 2002. – 751 с.
97. Чопей Р. С. Огляд підходів до аналізу тривалості виконання програмного коду [Текст] / Р. С. Чопей, Д. В. Федасюк. // Науково-технічний журнал "Електротехнічні та Комп’ютерні системи". – Одеса: Одеський національний політехнічний університет, 2017. – №26 (102). – С. 68 – 78.

98. Чопей, Р. С. Метод побудови засобу для автоматизованого тестування тривалості виконання програмного коду вбудованих систем розроблених з використанням KEIL µVISION [Текст] / Р. С. Чопей, Д. В. Федасюк // Електротехнічні та Комп'ютерні Системи: Теорія та Практика (ЕЛТЕКС – 2018): матер. Міжнар. наук.-практ. конф., 29 травня – 1 червня 2018, Одеса. – Одеса, 2018. – С. 34 – 40
99. Чуев Ю. В. Прогнозирование количественных характеристик процессов Файл формата PDF размером 186,32 МБ / Ю. В. Чуев, Ю. Б. Михайлов, В. И. Кузьмин., 1975. – 400 с.
100. Федасюк Д. В. Алгоритм побудови графу потоку керування за текстом програми мовою С [Текст] / Д. В. Федасюк, Р. С. Чопей // Науковий журнал Радіоелектроніка, інформатика, управління. – Запоріжжя: Запорізький національний технічний університет, 2018. № 2. – С. 154 – 161.
101. Windows ОС [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/windows.
102. Мова програмування C# [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://msdn.microsoft.com/uk-ua/vstudio/aa336809.aspx.
103. Чопей Р. С. Метод аналізу тривалості виконання програмного коду з урахуванням архітектури мікроконтролера / Р. С. Чопей, Д. В. Федасюк // Вісник Вінницького політехнічного інституту – Вінниця, 2018, №2(137). – С. 74 – 79.
104. About the Embedded Trace Macrocell [Електронний ресурс] // ARM Information Center – Режим доступу до ресурсу: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337e/CHDBGEED.html.
105. IEEE-ISTO 5001-2003 - Standard for a Global Embedded Processor Debug Interface // IEEE Org. – Режим доступу до ресурсу:

http://www.arbitrarytechnology.com/files/Download/ieee-5001%20nexus%20protocol.pdf.

106. Eclipse Integrated Development Environment (IDE) [Електронний ресурс] // Eclipse Fondation – Режим доступу до ресурсу: https://www.eclipse.org/ide/.
107. Intel MCS-51 Microcontroller Family User's Manual [Електронний ресурс] // Intel Corporation – Режим доступу до ресурсу: http://eeweb1.poly.edu/networks/specs/27238302.pdf
108. ADSP-21020 Analog Devices' SHARC® processor datasheet (Rev. D) [Електронний ресурс] // Analog Devices – Режим доступу до ресурсу: https://www.analog.com/media/en/technical-documentation/data-sheets/ADSP-21061_21061L.pdf
109. Atmel AVR Microcontrollers [Електронний ресурс] // Atmel Corporation – Режим доступу до ресурсу: https://www.microchip.com/wwwproducts/en/ATmega32
110. SPARC Series Processors ERC32 Documentation [Електронний ресурс] // Temic – Режим доступу до ресурсу: http://klabs.org/DEI/Processor/sparc/ERC32/ERC32_docs.htm

Додаток А. Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Fedasyuk D. A method of predicting the maintenance period of embedded systems for preventing breach of their time requirements / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Computing, Ukraine. 2018. – Vol. 17. – Issue 2, pp. 94–101. (Scopus; Google Scholar; Index Copernicus).

2. Fedasyuk D. A Model for Estimating Firmware Execution Time Taking Into Account Peripheral Behavior / D. Fedasyuk, T. Marusenkova, R. Chopey // International Journal of Intelligent Systems and Applications, Hong Kong. – 2018. – Vol. 10. – No. 6, pp 22-29. (Scopus; Google Scholar; Microsoft Academic Search; Stanford University Libraries).

3. Fedasyuk D. Determining the Influence of the External Factors on the Firmware Response Time / D. Fedasyuk, V. Gavrysh, T. Marusenkova, R. Chopey // Central European Researchers Journal, Slovakia. – 2018. – Vol. 4. – Issue 1, pp. 1–8.

4. Федасюк Д. В. Алгоритм побудови графу потоку керування за текстом програми мовою С [Текст] / Д. В. Федасюк, Р. С. Чопей // Науковий журнал Радіоелектроніка, інформатика, управління. – Запоріжжя: Запорізький національний технічний університет, 2018. № 2. – С. 154-161. (Thomson Reuters Web of Science; Google Scholar; eLIBRARY.RU; Index Copernicus).

5. Чопей Р. С. Огляд підходів до аналізу тривалості виконання програмного коду [Текст] / Р. С. Чопей, Д. В. Федасюк. // Науково-технічний журнал "Електротехнічні та Комп’ютерні системи". – Одеса: Одеський національний політехнічний університет, 2017. – №26 (102). – С. 68-78. (eLIBRARY.RU; Index Copernicus).

6. Чопей Р. С. Метод аналізу тривалості виконання програмного коду з урахуванням архітектури мікроконтролера [Текст] / Р. С. Чопей, Д. В. Федасюк // Вісник Вінницького політехнічного інституту – Вінниця, 2018, №2(137). – С. 74-79. (Index Copernicus; РИНЦ).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

7. Fedasyuk, D. Architecture of a Tool for Automated Testing the Worst-

Case Execution Time of Real-Time Embedded Systems' Firmware [Text]. / D. Fedasyuk, R. Chopey, B. Knysh // The experience of designing and application of CAD systems in microelectronics of the XIVth International conference CADSM'2017, February 21-25. – Lviv, 2017. – P. 278-282. (SCOPUS).

8. Chopey, R. A Model For Estimating Firmware Execution Time Based On Peripherals' Time Behavior [Text]. / R. Chopey, D. Fedasyuk, T. Marusenkova // Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering of the XIVth IEEE International Conference TCSET'2018, February 20-24. – Lviv, 2018. – P. 1179-1183. (SCOPUS).

9. Chopey R. The model of software execution time remote testing [Text]. / R. Chopey, B. Knysh, D. Fedasyuk // Computer Science and Engineering 2017 of the IXth International Conference of Young Scientists CSE'2017, November 22-24. – Lviv, 2017. – P. 398-402.

10. Чопей, Р. С. Метод побудови засобу для автоматизованого тестування тривалості виконання програмного коду вбудованих систем розроблених з використанням KEIL μVISION [Текст] / Р. С. Чопей, Д. В. Федасюк // Електротехнічні та Комп'ютерні Системи: Теорія та Практика (ЕЛТЕКС – 2018) : матер. Міжнар. наук.-практ. конф., 29 травня – 1 червня 2018, Одеса, 2018. – С. 34-40.

11. Чопей, Р.С. Проблеми тестування вбудованих систем жорсткого реального часу [Текст] / Р. С. Чопей, Д. В. Федасюк // Математичне і програмне забезпечення інтелектуальних систем (MPZIS-2016) : матер. XIV Міжнар. наук.-практ. конф., 16-18 листопада 2016, Дніпро. – Дніпро, 2016. – С. 243-245.

12. Чопей, Р.С. Тестування тривалості виконання програмного коду мікроконтролерних систем реального часу [Текст] / Р. С. Чопей, Д. В. Федасюк // Математичне і програмне забезпечення інтелектуальних систем (MPZIS-2017) : матер. XV Міжнар. наук.-практ. конф., 22-24 листопада 2017, Дніпро. – Дніпро, 2017. – С. 216-218.

Додаток Б. Акти впровадження результатів дисертації

Затверджую
Approved by
Директор-розпорядник компанії
Dinamica Generale S.p.A.
Managing Director of
Dinamica Generale S.p.A.

DINAMICA GENERALE S.p.A.
Via Modigliani n. 45
46026 Poggio Rusco (Mantova)
Tel. 0386.63 34 11 Fax 0386.51825
C.F. e P.I. 01599500202 - Iscr. REA MN 169392

Andrea Ghiraldi

/2018

АКТ
про впровадження результатів
дисертаційної роботи аспіранта кафедри програмного забезпечення
Національного університету «Львівська політехніка»
Чопея Ратібора Степановича

Statement of the use of research results
of thesis performed by Ratybor Chopey,
a postgraduate student in Lviv Polytechnic National University

Даний акт складений про те, що програмний засіб для тестування тривалості виконання програмного коду вбудованих систем «EXTT», розроблений під час виконання дисертаційного дослідження Чопея Ратібора Степановича на тему *«Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих системи»* впроваджено на Dinamica Generale S.p.A (м. Поджо Руско, Італія).

Програмний засіб «EXTT» дає можливість підвищити точність отриманих результатів на 3–5 %, а також зменшити тривалість тестування на 11–15%. Крім того, можливість здійснення прогнозування тривалості виконання програмного коду, дає можливість розробити план технічного обслуговування вбудованих систем, що забезпечує задану тривалість виконання програмного коду впродовж усього періоду експлуатації вбудованої системи.

Даний акт не є підставою для взаємних фінансових розрахунків.

The statement is drawn to certify that the software tool "EXTT" developed for testing the execution time of firmware running inside embedded systems as part of scientific research of Ratybor Chopey in his thesis "*Means of automated testing of specialized embedded systems firmware*" has been used in Dinamica Generale S.p.A (Poggio-Rusco, Italy).

The software «EXTT» allows us to enhance the accuracy of the obtained results by 3–5 % and decrease the duration of testing procedure by 11–15%. Besides, the possibility to forecast the execution time of firmware makes it possible to develop a maintenance plan for an embedded system, which would ensure the required firmware execution time during the whole life time of the embedded system.

The statement is not the subject of mutual financial arrangements.

Директор-розпорядник
Managing director

DINAMICA GENERALE S.p.A.
Via Mobradori n. 15
46025 Poggio-Rusco (Mantova)
Tel. 0386.52134 - Fax 0386.51623
C.F./P.I. 01...

Andrea Giraldi

ЕКОТЕСТ®
Засоби радіаційного контролю

ЗАТВЕРДЖУЮ
Директор-Генеральний конструктор
ПП "НВПП "Спаринг-Віст Центр"
_____ Ю.Б. Сторонський
"___" _____ 2018 р.

АКТ
про впровадження результатів дисертаційної роботи аспіранта кафедри програмного забезпечення Національного університету «Львівська політехніка» Чопея Ратібора Степановича

Даний акт складений про те, що на ПП "НВПП "Спаринг-Віст Центр" (м. Львів) впроваджені результати дисертаційного дослідження Чопея Ратібора Степановича на тему *«Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих системи»*, а саме метод формування плану тестування з урахуванням поведінки периферійних пристроїв.

Використання зазначеного методу в процесі тестування вбудованих систем, які розробляються ПП НВПП „Спаринг-Віст Центр" порівняно з іншими методами формування плану тестування забезпечило зменшення тривалості тестування на 11–15 %, що дало можливість підвищити ефективність процесу розроблення програмного забезпечення вбудованих систем.

Даний акт не є підставою для взаємних фінансових розрахунків.

ПП „НВПП „Спаринг-Віст Центр"
Україна, 79026, м. Львів, вул. Володимира Великого, 33
тел.: (032) 2421515, факс: (032) 2422015
e-mail: market@ecotest.ua, www.ecotest.ua

"ЗАТВЕРДЖУЮ"
Проректор з наукової роботи
Національного університету
"Львівська політехніка"
_____ проф. Н.І. Чухрай
"__" _____ 2018 р.

АКТ

про використання результатів дисертаційної роботи Чопея Ратібора Степановича на тему «Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем», поданої на здобуття наукового ступеня кандидата технічних наук.

Комісія у складі начальника науково-дослідної частини, к.т.н., доцента Жук Л. В., завідувача відділу науково-організаційного супроводу наукових досліджень, к.т.н. Лазько Г. В., заступника начальника планово-фінансової групи Чулой Т. М., завідувача кафедри програмного забезпечення, д.т.н., проф. Яковини В. С., цим актом підтверджує, що результати дисертаційної роботи Чопея Ратібора Степановича на тему «Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем» використані при виконанні науково-дослідної роботи "Розробка математичного забезпечення для програмного засобу аналізу функціональної безпечності та надійності програмно-апаратних систем відповідального призначення" (№ держреєстрації 0117U004458), а саме: розроблено метод прогнозування тривалості виконання програмного коду в програмно-апаратній системі з визначенням періодичності її технічного обслуговування.

Цей метод дає змогу автоматизовано проводити аналіз залежності тривалості виконання програмного коду від періодичності технічного обслуговування програмно-апаратної системи з урахуванням впливу на її компоненти зовнішніх і внутрішніх чинників. Використання запропонованого методу підвищує достовірність результату прогнозування та забезпечує заданий рівень надійності і безпечності експлуатації програмно-апаратної системи.

Начальник НДЧ, к.т.н., доц.	Жук Л.В.
Зав. відділу НОСНД, к.т.н.	Лазько Г.В.
Заст. начальника ПФГ	Чулой Т.М.
Зав. каф. ПЗ, д.т.н., проф.	Яковина В.С.
Науковий керівник НДР д.т.н., проф.	Волочій Б.Ю.

«ЗАТВЕРДЖУЮ»

Проректор з науково-педагогічної роботи

НУ «Львівська політехніка»

_____ доц. Давидчак О.Р.

«____»_____2018 р.

АКТ

про використання результатів кандидатської дисертаційної роботи
Чопея Ратібора Степановича
«Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем» у навчальному процесі
кафедри «Програмне забезпечення»

Даний акт складено комісією у складі:

д.т.н., проф. Медиковський М.О. – директор Інститути комп'ютерних наук та інформаційних технологій;

д.т.н., проф. Яковина В.С. – завідувач кафедри програмного забезпечення;

к.т.н., доц. Марусенкова Т. А. – доцент кафедри програмного забезпечення, лектор дисципліни;

про те, що в навчальному процесі кафедри програмного забезпечення використано результати кандидатської дисертаційної роботи «Засоби автоматизованого тестування спеціалізованого програмного забезпечення вбудованих систем», а саме для студентів напряму 6.050103 «Програмна інженерія» в лекційному курсі та практикумі дисципліни «Програмування мікроконтролерів».

Для студентів вивчення нового методу динамічного тестування тривалості виконання програмного коду відкриває можливість виконувати тестування вбудованої системи без внесення змін у її програмне чи апаратне забезпечення. При цьому побудова засобів тестування вбудованих систем згідно з розробленим методом дає змогу знизити часові витрати на їх розроблення та тестування.

Члени комісії _____ Медиковський М. О.

_____ Яковина В. С.

_____ Марусенкова Т. А.