УДК 004.054

МЕТОД ПОБУДОВИ ЗАСОБУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ ВБУДОВАНИХ СИСТЕМ РОЗРОБЛЕНИХ З ВИКОРИСТАННЯМ KEIL UVISION

Р. С. Чопей, Д. В. Федасюк

Національний університет «Львівська політехніка»

Анотація. *Розглянуто методи динамічного тестування тривалості виконання програмного забезпечення вбудованих систем. Представлено метод побудови засобу автоматизованого тестування, що дозволяє повністю контролювати процес виконання програми, а також проводити її тестування без використання додаткового спеціалізованого програмного чи апаратного забезпечення. Запропонований метод та алгоритм є універсальним для всіх вбудованих систем, що використовують мікроконтролери з ядром ARM, програмне забезпечення яких розроблене у середовищі Keil uVision.*

Ключові слова: *тестування вбудованих систем, вбудовані системи реального часу, CMSIS RTOS RTX, uVision Socket Interface.*

Вступ

На сьогоднішній день розвиток галузі вбудованих систем, зумовлено необхідністю використання спеціалізованої апаратури, що характеризується високою швидкодією, та низьким рівнем енергоспоживання [1]. Використання вбудованих систем у галузях, що є критичними з точки зору безпечності передбачає потребу у висуненні більш жорстких вимог до надійності та безпечності вбудованих систем та їх забезпечення на усіх етапах розробки. Одним зі шляхів забезпечення заданого рівня надійності є тестування тривалості виконання програмного коду, що є обов'язковим кроком при розробці вбудованих систем реально часу, адже правильність роботи програмного забезпечення залежить не лише від логічної правильності, а від часу за який отримано результат.

За останні десятиліття запропоновано багато методів для тестування тривалості виконання програмного коду, зокрема тестуванню найгіршого часу виконання програмного коду (англ. worst-case execution time, WCET). Ці методи можуть бути класифіковані згідно з процесом їх виконання: статичні методи та динамічні методи [2].

Статичні методи не передбачають виконання програми на реальному обладнанні чи симуляторі. У якості вхідних даних використовують програмний код, а також модель архітектури апаратного забезпечення. На основі аналізу можливих шляхів потоку керування визначають межі тривалості виконання програми.

Основним недоліком таких методів є складність створення адекватних моделей для низькорівневого аналізу, спрощення яких призводять до завищення отриманих результатів, що, водночас, призводить до надлишкового резервування ресурсів системи.

Динамічні методи базуються на вимірюванні тривалості виконання програмного коду на вбудованій системі або симуляторі. Виміряна (спостережувана) максимальна та мінімальна тривалість виконання фрагменту програми об'єднують для визначення тривалості виконання усього потоку.

Суттєвою перевагою над статичними методами є зменшення часових витрат за рахунок відмови від розроблення та аналізу моделей поведінки апаратного забезпечення. Однак при їх використанні виникає ряд проблем, зокрема: складність спостереження та контроль за виконанням тестів на вбудованій системі, без внесення змін у її програмне чи апаратне забезпечення [3].

Усе сказане вище приводить до ідеї розроблення методу тестування тривалості виконання програмного коду мікроконтролерних вбудованих систем реального часу, що дає змогу повністю контролювати стан вбудованої системи та процес тестування, без внесення буд-яких змін у програмне та апаратне забезпечення.

1. Огляд методів тестування тривалості виконання програмного коду вбудованих систем

У цьому розділі описані загальні методи тестування тривалості виконання програмного коду їх переваги та недоліки.

1.1 Тестування тривалості виконання програмного коду з допомогою інтегрування додаткового програмного забезпечення

Загальна ідея цього підходу передбачає інтегрування у програмне забезпечення вбудованої системи, додаткового програмного забезпечення (фреймворк), що контролює виконання програми вбудованої системи, здійснює вимірювання тривалості виконання функцій та потоків, та надсилає отримані дані до комп'ютера (рис 1.).

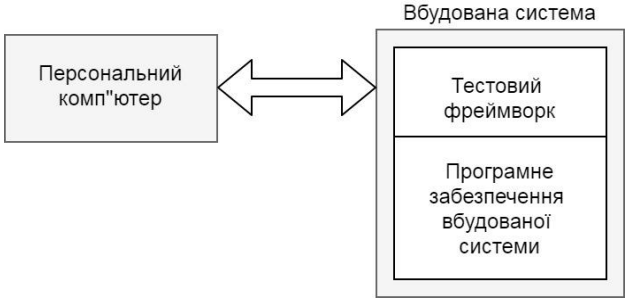

Рис. 1. Блок-схема системи тестування з використанням додаткового програмного забезпечення

У [4] запропоновано архітектуру засобу автоматизованого тестування найгіршої тривалості виконання програмного коду, що потребує внесення у програмний код вбудованої системи, фреймворка для вимірювання тривалості виконання фрагменту коду чи потоку. Вимірювання тривалості виконання відбуваються шляхом внесення у текст програми контрольних точок, при досягнені яких вбудована система запускає/зупиняє таймер для вимірювання тривалості виконання програмного коду та надсилає результати вимірювання до персонального комп'ютера.

У [5] представлено проблеми, що виникають при аналізі найгіршої тривалості виконання програмного коду. Також запропоновано архітектуру засобу для автоматизованого найгіршої тривалості виконання програмного коду. Ідея цього підходу передбачає створення системи для тестування, що працює за принципом Host → Target. Host програмний застосунок, що працює на персональному комп'ютері, та призначений для генерування тестових даних, надсилання їх до вбудованої системи (Target), а також формування звіті про результати тестування. У свою чергу вбудована система містить окрім основного програмного забезпечення, додаткове, яке необхідне для виконання тестування. Додаткове програмне забезпечення контролює вбудовану систему, виконує тестування, та надсилає результати тестування до персонального комп'ютера.

Застосування додаткового програмного забезпечення для тестування тривалості виконання програмного коду є одним з найбільш розповсюджених підходів. Однак такі підходи характеризуються рядом недоліків та проблем, що є вкрай важливі при вимірюванні тривалості виконання програмного коду:

– Зміна функціоналу. Інтегрування додаткового програмного коду у програмне забезпечення вбудованої системи певною мірою змінює це програмне забезпечення та його цільове призначення загалом. Відтак, окрім зменшення адекватності отриманих результатів вимірювання, збільшується ймовірність появи помилок, що викликана внесеним додатковим програмним забезпеченням.

– Використання додаткових апаратних ресурсів. Використання апаратного таймера мікроконтролера та одного з вбудованих інтерфейсів зв'язку, для проведення вимірювання тривалості виконання програмного коду, може створити додаткові складнощі у випадку, якщо існуюче програмне забезпечення використовує усі доступні апаратні ресурси.

– Особливі вимоги до архітектури програмного забезпечення вбудованих систем. Використання підходу описаного у [5] передбачає необхідність побудови програмного забезпечення вбудованих систем за архітектурою Superloop [6], що дає змогу потрапити у будь-яку гілку програми змінюючи значення однієї або декількох глобальних змінних.

1.2 Тестування тривалості виконання програмного коду з використанням симуляторів вбудованої системи

Симулятори процесорів використовуються для часового аналізу, що замінює реальне обладнання.

У [7] представлено методологію розкладання програмного забезпечення на окремі, ізольовані компоненти та їх аналіз. Для аналізу контекстної залежності (аналізу кеш-пам'яті) використаний симулятор процесора, що розроблений авторами. Отримані результати є близькими до раніше опублікованих результатів, одержаних з використанням статичних методів. Тому автори стверджують, що розроблена методологія є безпечною та може застосовуватися для аналізу часу виконання вбудованих систем жорсткого реального часу.

Недоліками використання симуляторів є:

– Висока складність. Процес розробки моделей часової поведінки процесора та зовнішніх периферійних пристроїв є дуже складним та три-

валим. Спрощення цих моделей призведе до втрати точності, та непридатності отриманих результатів.

1.3 Тестування тривалості виконання програмного коду з використанням апаратного трасування

Розвиток цих підходів, що базуються на вимірюванні часу виконання програмного коду з допомогою апаратного трасування, зумовлений розвитком модулів для відлагодження програмного забезпечення, які інтегровані у мікроконтролер. Окрім того, їх використання дає повний контроль над вбудованою системою з використанням лише програматора, що використовується у процесі розробки програмного забезпечення.

У роботі [8] розглянуто архітектуру засобу для автоматизованого тестування вбудованих систем, що передбачає проведення тестування безпосередньо на вбудованій системі через середовище розробки програмного забезпечення Eclipse IDE. Середовище розробки зв'язується із вбудованою системою через інтерфейс відлагодження.

Розроблений засіб погано адаптований для визначення WCET програмного коду. Окрім цього, цей підхід не передбачає можливість тестування гілок програми, вхід у які спричинений відмовами апаратного забезпечення та тимчасовими порушеннями працездатності, а відтак цей засіб не може забезпечити повного покриття програмного коду тестовими сценаріями.

Загальним недоліком підходів, що базуються на апаратному трасування є складність їх реалізації. Основною перевагою над іншими методами тестування є повний контроль над вбудованою системою, та процесом тестування.

2. Метод побудови засобу автоматизованого тестування тривалості виконання програмного коду

Тестування тривалості виконання програмного коду безпосередньо на мікроконтролерній вбудованій системі вимагає застосування спеціалізованого програмного засобу, що дає змогу надсилати та отримувати данні до\з вбудованої системи, повністю її контролювати без застосування додаткового програмного чи апаратного забезпечення. Такий підхід не змінює вбудовану систему, а відтак отримані результати будуть достовірними.

У цьому розділі наведено метод побудови програмного модуля тестування тривалості виконання програмного коду, деталі його реалізації, алгоритм роботи та отримані результати.

2.1 Метод побудови програмного модуля тестування тривалості виконання програмного коду.

Метод побудови програмного модуля тестування тривалості виконання програмного коду вбудованих систем зображено на рис.2.

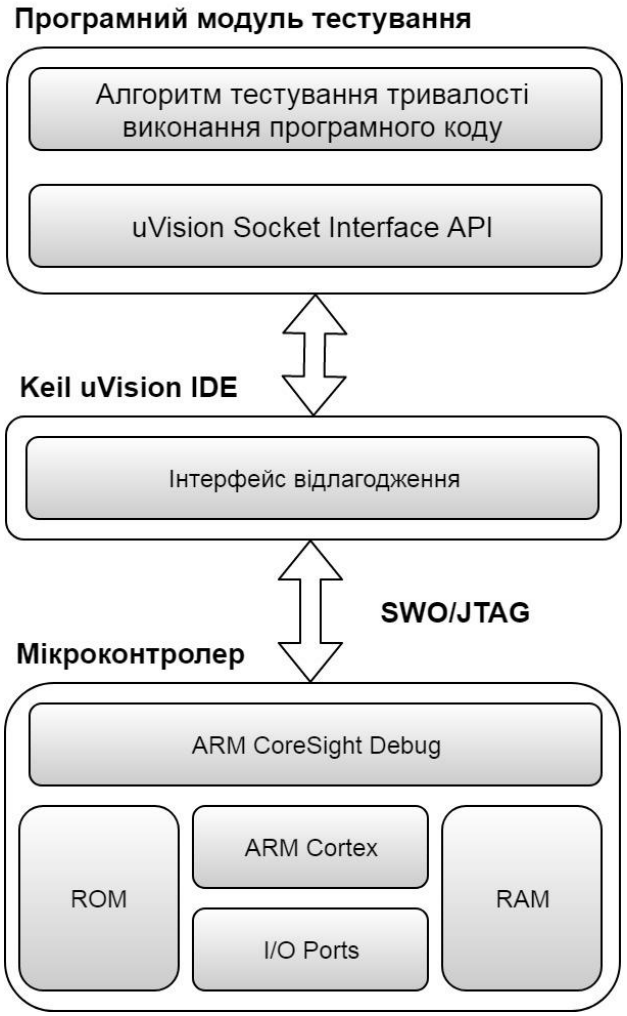

Рис. 2. Блок-схема системи тестування з використанням інтерфейсу uVision Socket

Програмний модуль тестування включає у собі алгоритм тестування, що передбачає вимірювання тривалості виконання програмного коду, а також uVision Socket Interface API (UVSOCK). UVSOCK – це інтерфейс, що дозволяє керувати і відстежувати стан середовища програмування Keil uVision IDE, через сторонні застосунки [9], а відтак повністю контролювати та керувати вбудовану системою в процесі виконання програми у режимі відлагодження. Тестування програмного забезпечення вбудованих систем, у режимі відлагодження, не впливає швидкість виконання програми, адже мікроконтролери, що побудовані на базі ядра ARM містять модуль відлагодження ARM CoreSight Debug[10], який є незалежним від основного ядра.

Запропонований метод побудови програмного модулю для тестування дає змогу знизити часові витрати на реалізацію протоколу обміну даними між модулем відлагодження ARM CoreSight Debug та програмним модулем тестування.

2.2 Алгоритм тестування тривалості виконання програмного коду вбудованих систем

Розроблений алгоритм передбачає, що час виконання програмного коду вимірюється за допомогою двох точок зупинки, одна встановлюється на початку функції(потоку), що тестується, а інша - наприкінці. Тривалість проходження між двома точками зупинки рівна тривалості виконання програмного коду, що повинна бути вимірюною. Алгоритм тестування представлений на рис. 3.

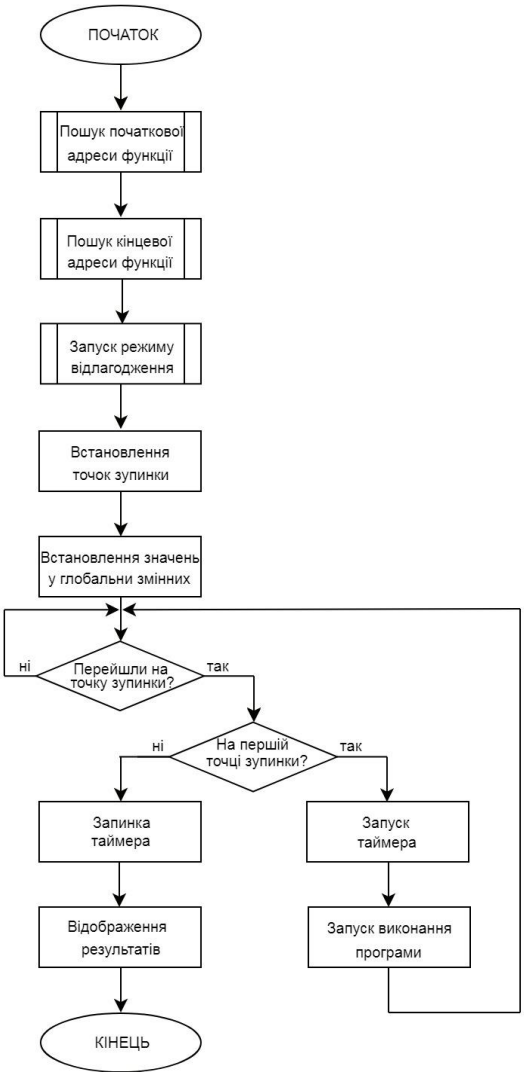

Рис. 3. Алгоритм тестування тривалості виконання програмного коду

Крок 1. Пошук початкової адреси функції, що підлягає тестуванню, за якою розміщується функція у пам'яті програм. На цьому кроці проводиться 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 та адреси за якими вони розміщення у оперативній пам'яті. У випадку не виявлення функції у map-файлі алгоритм надсилає спеціальне повідомлення до користувацького інтерфейсу.

Крок 2. Пошук кінцевої адреси функції. Цей крок передбачає проведення синтаксичного аналізу listing файлу, що генерується в процесі компіляції. 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 uVision, та початок виконання програми.

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

Крок 5. Встановлення у глобальних змінних значення, що забезпечать перехід до виконання функції, яка підлягає тестуванню.

Крок 6. При потраплянні у колбек функцію зупинки виконання програми у встановленій точці, запускаємо таймер, що використовується для вимірювання тривалості виконання програми.

Крок 7. Надсилання команду продовження виконання програми.

Крок 8. При повторному потраплянні у колбек функцію зупинки виконання програми, запиняємо таймер. Та відображаємо отримані результати.

2.3 Експерименти

Для дослідження запропонованого методу побудови засобу для автоматизованого тестування, запропонований алгоритм реалізований у вигляді окремого програмного модуля з використанням мови C#.

Функція обрана для тестування реалізовувала почергову зміну логічного рівня на піні мікроконтролера, з інтервалом 1 с. Функція затримки реалізована з використанням апаратного таймера, що забезпечує обрану тривалість. Текст програми наведено нижче:

```
void SysTick_Handler(void)
{
      ticks_delay++;
}

void delay(uint32_t milliseconds)
{
```

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

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

У якості опорного значення, проведено вимірювання періоду зміни логічного рівня на піні мікроконтролера, використовуючи цифровий осцилографа. Вимірювання тривалості виконання обраної функції запропонованим алгоритмом, проведено 100 разів. Середня тривалість виконання функції ChangeLevel отримана з використання запропонованого підходу рівна 2176,6 мс, що на 176,6 мс більша за значення отримане при вимірюванні цифровим осцилографом. Похибка вимірювання запропонованим алгоритмом є методичною, і зумовлена затримкою при обміні повідомленнями між розробленим програмним модулем та вбудованою системою, а відтак її значення легко компенсувати.

Запропонований метод, побудови засобів тестування вбудованих систем, використовувався при створення засобу віддаленого тестування тривалості виконання програмного коду, що експериментально підтвердив ефективність його застосування [10].

Висновки

Згідно з запропонованим методом та алгоритм для тестування тривалості виконання програмного забезпечення вбудованих систем досліджено можливість тестування тривалості виконання програмного коду мікроконтролерних вбудованих системи.

У ході виконання експериментів виявлено вплив затримок при обміні повідомленнями між вбудованою системою та програмним модулем. Виявлено, що значення похибки є постійним та залежить від швидкодії комп'ютера на якому виконується алгоритм тестування, а також швидкості на якій працює програматор, через який здійснюється обмін повідомленнями. Це дає змогу компенсувати значення методичної похибки, та підвищити точність вимірювання тривалості виконання програмного коду.

Запропонований метод побудови засобу автоматизованого тестування є універсальним для усіх мікроконтролерів з архітектурою ARM, програмне забезпечення, яких розроблене у середовищі Keil uVision. Використання можливостей вбудованого у мікроконтролер модуля відлагодження ARM CoreSight Debug, дає змогу повністю контролювати виконання програми, та хід тестування без використання додаткового програмного та апаратного забезпечення.

Представлений метод побудови засобу автоматизованого може використовуватись для функціонального та нефункціонального тестування. Окрім того наявність у uVision Socket Interface інтерфейсу TCP/IP, дає змогу створити алгоритмів для віддаленого тестування вбудованих систем.

Перспективи подальших досліджень полягають у інтегруванні запропонованого алгоритму у засіб автоматизованого тестування тривалості виконання програмного коду вбудованих систем.

Список використаної літератури

1. Embedded Software: Know It All [Text] / [J. Labrosse, J. Ganssle, R. Oshana, C. Walls, K. Curtis, J. Andrews, D. Katz, R. Gentile, K. Hyder, B. Perrin]. – Burlington : Newnes, 2008. – 745 p. ISBN-13: 978-0750685832

2. Чопей, Р. Огляд підходів до аналізу тривалості виконання програмного коду [Текст] / Р. С. Чопей, Д. В. Федасюк. // Науково-технічний журнал Електротехнічні та Комп`ютерні системи. – 2017. – № 26 (102). – С. 68–78. DOI: 10.15276/eltecs.26.102.2017.8.

3. Lindstrom, B. Six Issues in Testing Event Triggered Systems [Electronic Resource]. – Access Mode : https://pdfs.semanticscholar.org/e2e0/7660fc107bcf73f0fbdb29618548032db95e.pdf.

4. Kirner, R. The WCET Analysis Tool CalcWcet167 [Text] / R. Kirner // Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies : 5th International Symposium, Heraklion, October 2012 : proceedings. – Heraklion, ISoLA, 2012. – P. 158 –172. DOI: 10.1007/978-3-642-34032-1_17.

5. Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware [Text] / [D. Fedasyuk, R. Chopey and B. Knysh] // The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM) : 14th International Conference, 21-25 February 2017 : proceedings. – Lviv, 2017. – P. 278–282.DOI: 10.1109/CADSM.2017.7916134.

6. Programming Embedded Systems II [Text] / [M. J. Pont]. – Westcotes, 2004, 287 p.

7. Structured Testing of Worst-Case Execution Time Analysis Tools [Text] / [J. Engblom, F. Stappert, and A. Ermedahl] // 21st Real-Time System

Symposium (RTSS/WIP'00), 27-30 November 2000 : proceedings. – Orlando, 2000. – P. 154–163.

8. CAST: Automating Software Tests for Ebedded Systems [Text] / [M. Wahler, E. Ferranti, R. Steiger, R. Jain et al.] // IEEE Fifth International Conference on Software Testing, Verification and Validation, 17-21 April 2012 : proceedings. – Montreal : IEEE, 2012, – P. 123 –133. DOI: 10.1109/ICST.2012.126

9. Application Note 198: Using the uVision Socket Interface r [Electronic Resource]. – Access Mode: http://www.keil.com/appnotes/docs/apnt_198.asp.

10. The model of software execution time remote testing [Text] / [R. Chopey, B. Knysh and D. Fedasyuk] // The 11th International Conference of Young Scientists «Computer Science and Engineering 2017» (CSE'2017), 2017 : proceedings. – Lviv, 2017. – P. 398–402.

References

1. J. Labrosse, (2008) Embedded software. Elsevier / Newnes, Burlinton,Mass., U.S.A. ISBN-13:978-0750685832

2. R. Chopey, D. Fedasyuk, (2017) "Overview of approaches to the execution time analysis of programs [Oglyad pidhodiv do analizu truvalosti vikonannya programnogo kodu]", Scientific and Technical Journal Electrotechnic and Computer Systems, no. 26(102), pp. 68–78. DOI: 10.15276/eltecs.26.102.2017.8.

3. Lindstrom, B., Six Issues in Testing Event Triggered Systems – Access Mode : https://pdfs.semanticscholar.org/e2e0/7660fc107bcf73f0fbdb29618548032db95e.pdf.

4. R. Kirner, (2012) "The WCET Analysis Tool CalcWcet167", Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, pp. 158 –172, DOI: 10.1007/978-3-642-34032-1_17.

5. D. Fedasyuk, R. Chopey and B. Knysh, (2017) "Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware", 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM-2017), pp. 278-282, DOI: 10.1109/CADSM.2017.7916134

6. Pont, M. J, (2004) Programming Embedded Systems II / Westcotes., UK, 287 pp. J. Engblom, F. Stappert, and A. Ermedahl(2000), "Structured Testing of Worst-Case Execution Time Analysis Tools", 21st Real-Time System Symposium (RTSS/WIP'00)

7. J. Engblom, F. Stappert, and A. Ermedahl, (2000) "Structured Testing of Worst-Case Execution Time Analysis Tools", 21st Real-Time System Symposium (RTSS/WIP'00).

8. M. Wahler, E. Ferranti, R. Steiger, R. Jain and K. Nagy, (2012) "CAST: Automating Software Tests for Embedded Systems", IEEE Fifth International Conference on Software Testing, Verification and Validation, DOI: 10.1109/ICST.2012.126

9. Application Note 198: Using the uVision Socket Interface – Access Mode : http://www.keil.com/appnotes/docs/apnt_198.asp. [Accessed: 29- Sep- 2017].

10. R. Chopey, B. Knysh, and D. Fedasyuk, (2017) "The model of software execution time remote testing", in Proc. of the IXth International Conference of Young Scientists «Computer Science and Engineering 2017» (CSE'2017), Lviv, pp. 398–402.

THE METHOD OF CONSTRUCTION OF THE MEANS FOR EXECUTION TIME TESTING THE EMBEDDED SYSTEMS THAT ARE DEVELOPED IN KEIL UVISION

R. S. Chopey, D. V. Fedasyuk

Lviv Polytechnic National University

Abstract. *We've considered the common dynamic methods of the firmware execution time testing. The authors proposed the method for the construction of the means for firmware execution time testing. This method based on the uVision Socket Interface API that give us the possibility to control the integrated development environment Keil uVision from external applications. As a result, we have the full control of the firmware executing and testing process, without using additional firmware or hardware.*

Upon the obtained research results, one may conclude that using the proposed method for construction of the means for firmware testing would allow us to:

– eliminate development process of the communication protocol between embedded system in debug mode and application for testing, which will reduce the total development time of the application for testing the embedded systems.

– to increase the accuracy of the obtained results due to the fact that measurements are conducted on real hardware;

The performed investigations showed that proposed method for the construction of the means for firmware testing can be used during constructing the application for functional and non functional testing. Experimentally proved that the proposed algorithm could be used for execution time testing. The investigations into the relative measurement error has proved that the error is constant and depended for the performance of PC, and speed of data exchanging between PC application and embedded system. The value of the measurement error could be calculated in a easy way, and compensate it. This allows you to get a high degree of accuracy in measuring of the firmware execution time, that is critical for hard real-time embedded systems.

Keywords: *embedded systems testing, embedded real-time systems, CMSIS RTOS RTX, uVision Socket Interface.*

МЕТОД ПОСТРОЕНИЯ СРЕДСТВА ДЛЯ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ПРОДОЛЖИТЕЛЬНОСТИ ВЫПОЛНЕНИЯ ПРОГРАММНОГО КОДА ВСТРАИВАЕМЫХ СИСТЕМ РАЗРАБОТАНЫХ С ИСПОЛЬЗОВАНИЕМ KEIL UVISION

Р. С. Чопей, Д. В. Федасюк

Национальный университет «Львовская политехника»

Аннотация. *Рассмотрены методы тестирования продолжительности выполнения программного кода встроенных систем. Представлен метод построения средства автоматизированного тестирования, позволяет полностью контролировать процесс выполнения программы, а также проводить ее тестирование без использования, дополнительного специализированного программного или аппаратного обеспечения. Предложенный метод и алгоритм является универсальным для всех встроенных систем, использующих микроконтроллеры с ядром ARM, программное обеспечение которых разработано в среде разработки Keil uVision.*

Ключевые слова: *тестирования встроенных систем, встроенные системы реального времени, CMSIS RTOS RTX, uVision Socket Interface.*

Отримано 14.03.2018

Чопей Ратібор Степанович, аспірант кафедри програмного забезпечення Національного університету «Львівська політехніка». вул. С. Бандери 28а, Львів, Україна, E-mail: chopey.ratybor@gmail.com, тел. +38-093-718-74-11

Ratybor Chopey, PhD student of the software department of Lviv Polytechnic National University, Lviv Polytechnic National University, S. Bandera str., 28a, Lviv, Ukraine, E-mail: chopey.ratybor@gmail.com, phone +38-093-718-74-11

ORCID ID: 0000-0002-0782-2336

Федасюк Дмитро Васильович, доктор технічних наук, професор, проректор Національного університету «Львівська політехніка». вул. С. Бандери 12, Львів, Україна, E-mail: dmytro.v.fedasyuk@lpnu.ua, тел. +38-032-258-20-50

Dmytro Fedasyuk, Dr. of Science, Professor, Vice-Rector of Lviv Polytechnic National University, Lviv Polytechnic National University, S. Bandera str., 12, Lviv, Ukraine, E-mail: dmytro.v.fedasyuk@lpnu.ua, phone +38-032-258-20-50

ORCID ID: 0000-0003-3552-7454