УДК 004.054

ОГЛЯД ПІДХОДІВ ДО АНАЛІЗУ ТРИВАЛОСТІ ВИКОНАННЯ ПРОГРАМНОГО КОДУ

Р. С. Чопей, Д. В. Федасюк

Національний університет «Львівська політехніка»

Анотація. *Розглянуто загальні проблеми, які виникають при аналізі тривалості виконання програмного коду, визначено проблеми, актуальні при аналізі тривалості виконання програмного коду мікроконтролерних вбудованих систем. Здійснено огляд методів статичного та динамічного аналізу тривалості виконання програмного коду. Доведено необхідність розроблення програмного засобу для часового аналізу з використанням інтерфейсів трасування, що дасть можливість повністю контролювати вбудовану систему та процес тестування тривалості виконання програмного коду без внесення змін у її програмне та апаратне забезпечення.*

Ключові слова: *аналіз тривалості виконання, вбудовані системи реального часу, CMSIS RTOS RTX, найгірша тривалість виконання, найкраща тривалість виконання.*

Вступ

Сучасний етап розвитку вбудованих систем (ВС) характеризується розширенням галузей застосування [1]. Стрімкий розвиток ВС зумовлений необхідністю створення спеціалізованої апаратури, яка характеризується високою швидкодією, низьким рівнем енергоспоживання та невеликими габаритами.

Більшість, якщо не усі, ВС є системами «реального часу». Система реального часу – це система, у якій правильність одержаних результатів залежить не лише від логічної коректності програми, але й від проміжку часу, за який вони отримані [2]. Залежно від наслідків, спричинених порушеннями часових вимог, вбудовані системи реального часу прийнято поділяти на системи «м'якого» та «жорсткого» реального часу [3]. Системи «м'якого» реального часу – це системи, у яких порушення часових вимог призведе до зниження якості роботи системи. Системи «жорсткого» реального часу – це системи, у яких порушення часових вимог є рівнозначним відмові системи, а відтак може призвести до катастрофічних наслідків. Саме це передбачає потребу висунення більш жорстких вимог до надійності та безпечності ВС та їх забезпечення на усіх етапах розробки.

Одним зі шляхів забезпечення заданого рівня надійності є аналіз тривалості виконання програмного коду, що є ключем до прогнозованої роботи системи, а відтак і до правильності її роботи.

Метою цієї роботи є аналіз методів визначення найгіршого часу виконання програмного коду, що можуть застосовуватись для мікроконтролерних вбудованих систем реального часу, та забезпечують високу точність отриманих результатів.

1. Аналіз тривалості виконання програмного коду

У цьому розділі описано проблеми, які роблять аналіз тривалості виконання програмного коду складною та актуальною темою досліджень. Представлена декомпозиція проблеми на завдання, та висновки щодо можливих шляхів їх вирішення.

1.1 Проблеми, що виникають при аналізі тривалості виконання програмного коду

Аналіз тривалості виконання програмного коду – це будь-який структурований метод чи інструмент, що застосовується для оцінювання тривалості виконання фрагменту програми. Одним із напрямів такого аналізу є аналіз найгіршої тривалості виконання програмного коду. Згідно з визначенням, найгірша тривалість виконання програмного коду (англ. worst-case execution time, WCET) – це максимальний час, що необхідний для виконання: певного фрагменту коду, в заданому контексті, на заданому апаратному забезпеченні. Окрім цього, аналіз тривалості виконання застосовується для визначення найкращої тривалості та середнього часу виконання програмного коду. Найкраща тривалість виконання програми (англ. best-case execution time, BCET) – це мінімальний час, що необхідний для виконання: певного фрагменту коду, в заданому контексті, на заданому апаратному забезпеченні. Середній час виконання програми (англ. average-case execution time, ACET) – перебуває в межах між найгіршою та найкращою тривалістю вико-

нання програми і залежить від розподілу часу виконання програми.

Аналіз тривалості виконання програмного коду супроводжується рядом проблем: аналіз потоку даних, залежність тривалості виконання програмного коду від контексту, часові аномалії планувальника операційних систем, затримки, спричинені апаратним забезпеченням. Розглянемо кожну з них дещо детальніше.

– Аналіз потоку даних. Потік або функція, що підлягає аналізу, досягає WCET на одному або декількох шляхах виконання. Вхідні дані та стан системи (значення у внутрішніх регістрах, дані, що знаходяться у кеш-пам'яті, стан апаратного забезпечення) є невідомі, а відтак, їх визначення є складним або неможливим.

Первинною проблемою є потреба побудови графу потоку керування з тексту програми або машинного коду. Наявність динамічних переходів та викликів функції створюють окрему проблему при аналізі графу потоку керування та визначенні можливого шляху виконання програми. Окрім цього, окремим завданням при аналізі графу потоку керування є визначення максимальної кількості ітерацій циклу та глибини рекурсії, витрати часу на виконання яких є найбільшими.

– Залежність тривалості виконання програмного коду від контексту. Ранні роботи, присвячені проблемі часового аналізу, передбачали контекстну незалежність, тобто тривалість виконання для окремих функцій не залежить від історії виконання. Виходячи з цього, структурний підхід [4] передбачав додавання показників WCET послідовних фрагментів коду. Якщо потік виконує послідовно два фрагменти коду A та B, тоді WCET для фрагменту коду B визначається, як сума WCET фрагмента коду A (wtA) та фрагмента коду B(wtB). Загальний WCET потоку визначається:

$$A + B = wtA + (wtA + wtB)$$

Зміна архітектури сучасних процесорів, зокрема поява кеш-пам'яті призвела до контекстної залежності тривалості виконання. Тобто тривалість виконання окремих інструкцій може змінюватись на декілька порядків залежно від стану процесора, на якому вони виконуються.

Усе зазначене вище призвело до появи додаткового виду аналізу – поведінки процесора. Цей вид аналізу передбачає збір інформації про поведінку процесора при виконанні фрагментів програми, що впливають на тривалість виконання. Аналіз зібраних даних про стан внутрішніх регістрів і кеш-пам'яті дає змогу визначити межі тривалості виконання команд та базових блоків програми.

– Часові аномалії планувальника операційних систем є одним з видів часових аномалій, вперше опубліковані у [5]. Аномальна поведінка планувальника операційних систем виникає у випадках, коли незалежні інструкції захоплюють спільні апаратні ресурси, що призводить до зміни тривалості виконання команд. Також часові аномалії порушують припущення, що для визначення WCET потоку достатньо обрати найгірші випадки виконання кожної інструкції потоку. Існування часових аномалій ставить під сумнів придатність методу аналізу найгіршого часу виконання, наведеного у роботі [6], що у подальшому може призвести до небезпечної поведінки системи.

– Затримки, спричинені апаратним забезпеченням, на якому виконується програма, є однією з найважливіших проблем.

Складність аналізу поведінки процесора та внутрішньої пам'яті зростає з розвитком його архітектури, та доповнюється потребою врахування поведінки інших компонент вбудованої системи: давачів, виконавчих пристроїв, зовнішньої пам'яті. Зовнішні компоненти мають більший вплив на тривалість виконання програм у випадках коли програма виконується у відповідь на зовнішню подію. Окрім цього, часові затримки апаратного забезпечення є недетермінованими, адже зумовлені процесом старіння та впливом зовнішніх факторів. Тому їх аналіз вимагає застосування доволі складних методів.

1.2 Актуальні проблеми для мікроконтролерних вбудованих систем

На сьогодні наведені проблеми, що виникають при аналізі тривалості виконання програмного коду, частково вирішені для того чи іншого класу систем. Також деякі з наведених проблем є неактуальними для певних класів систем. Зокрема, проблема побудови динамічного графу потоку керування неактуальна для вбудованих систем жорсткого реального часу, програмне забезпечення яких розробляється мовою С з використанням структурного методу, а відтак граф потоку керування для такого програмного забезпечення є статичними. Окрім цього, визначення глибини рекурсії при аналізі графу потоку керування є неактуальним, якщо програмне забезпечення розробляється згідно стандартом якості – MISRA C-2004 [7], що забороняє використання рекурсії.

Проблема часових аномалій планувальника операційних систем втрачає актуальність при використанні операційних систем реального ча-

су, зокрема CMSIS RTOS RTX [8]. Однак, використання таких операційних систем з циклічним запуском завдань для їх квазіпаралельного виконання ускладнює процес часового аналізу.

Подальший огляд методів аналізу тривалості виконання буде орієнтований на можливість їх застосування для мікроконтролерних вбудованих систем жорсткого реального часу.

2. Методи часового аналізу

За останні десятиліття запропоновано багато методів для аналізу тривалості виконання програмного коду, зокрема аналізу WCET. Ці методи можуть бути класифіковані згідно з процесом їх виконання: динамічні методи та статичні методи.

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.

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.

2.1 Динамічні методи аналізу тривалості виконання програмного коду

Класичним методом отримання інформації про тривалість виконання програми є її вимірювання з різними вхідними даними. Пошук вхідних даних, що приводить до найгіршої тривалості виконання, є дуже складним. Тому гарантувати те, що він був знайдений, без статичної перевірки неможливо. Часто вимірювання виявляється єдиним доступним методом для програміста. Окрім цього, з точки зору апаратного забезпечення метод вимірювання має потенційну перевагу у тому, що програма виконується на реальному апаратному забезпеченні. Це дає змогу уникнути потреби побудови моделі апаратного забезпечення, які необхідні для методів статичного аналізу.

Розроблені методи вимірювання тривалості виконання програми, що використовуються на практиці, є такими: з допомогою осцилографа та логічного аналізатора; з допомогою апаратного трасування, з допомогою інтегрування додаткового ПЗ, з використанням симуляторів вбудованої системи. Розглянемо кожен з них дещо детальніше:

– **Вимірювання тривалості виконання програмного коду з допомогою осцилографа та логічного аналізатора.** Ці методи розглядають видиму поведінку системи під час її роботи без впливу на неї. Осцилограф чи логічний аналізатор застосовується для спостереження за зміною логічного рівня сигналу на виводі процесора, при виконанні певного сегмента програми. Отримані дані використовуються для прогнозування тривалості виконання [9, 10]. Недоліком цього методу є можливість вимірювання тривалості виконання програмного коду, що відповідає за видиму поведінку системи. Окрім цього, точність вимірювання безпосередньо залежить від обладнання, що використовується.

– **Вимірювання тривалості виконання програмного коду з допомогою апаратного трасування.** Трасування роботи вбудованої системи можливе при використанні вбудованих у процесор модулів відлагодження. Найбільш відомими прикладами таких інтерфейсів є ARM Embedded Trace Macrocell (ETM), Joint Test Action Group (JTAG), а також інтерфейс для відлагодження Nexus debug interfaces (NDI).

У роботі [11] 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.

– **Вимірювання тривалості виконання програмного коду з допомогою інтегрування додаткового програмного забезпечення** (фреймворків), є одним з найбільш розповсюджених підходів.

У роботі [12] подано архітектуру засобу автоматизованого тестування найгіршої тривалості виконання програми, що потребує внесення у програмний код вбудованої системи, фреймворка для виконання вимірювання тривалості виконання фрагменту коду чи потоку.

У [13] представлено проблеми, що виникають при аналізі найгіршої тривалості виконання програми, а також архітектуру засобу для автоматизованого вимірювання тривалості виконання функцій та потоків. Представлена архітектура

засобу передбачає інтегрування у програмне забезпечення вбудованої системи додаткового потоку тестування, що приймає дані з комп'ютера та надсилає результати тестування. Вимірювання тривалості виконання програми здійснюється з допомогою апаратного таймера вбудованої системи.

Загальним недоліком інтегрування додаткового програмного коду у програмне забезпечення вбудованої системи є те, що певною мірою воно 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.

– **Вимірювання тривалості виконання програмного коду з використанням симуляторів вбудованої системи.** Симулятори процесорів використовуються для часового аналізу, що замінює реальне обладнання. Розробка і перевірка симуляторів є дуже складним і трудомістким завданням, тому на практиці вони не можуть гарантувати точність отриманих даних.

У [14] представлено методологію розкладання програмного забезпечення на окремі, ізольовані компоненти та їх аналіз. Для аналізу контекстної залежності (аналізу кеш-пам'яті) використаний симулятор процесора, 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.

Недоліком використання симуляторів є складність та значна тривалість процесу розробки моделей процесора та апаратного забезпечення вбудованої системи. Спрощення моделей для збільшення швидкості їх створення призведе до втрати точності, та непридатності отриманих результатів.

2.2 Статичні методи аналізу тривалості виконання програмного коду

Робота статичного аналізатора WCET вважається правильною, якщо прогнозована тривалість виконання програми більша або рівна фактичному значенню WCET програмного коду. Зображений на рис.1 розподіл відображає актуальність отриманих даних, при застосуванні статичних та динамічних методів аналізу найгіршої тривалості виконання (WCET) та найкращої тривалості виконання програми (BCET).

Незважаючи на завищені результати часового аналізу, статичним методам надають більшу перевагу і застосовують їх при аналізі тривалості виконання програми вбудованих систем м'якого реального часу, для яких визначення середньої тривалості виконання є більш пріоритетним. Однак ці методи застосовуються і для аналізу WCET.

Рис. 1. Розподіл часу виконання програми [15]

Зазвичай статичний аналіз ділиться на три етапи: аналіз потоку виконання програми; аналіз функцій низького рівня; розрахунок найгіршої тривалості виконання програмного коду. Розглянемо кожен з них дещо детальніше.

– **Етап 1. Аналіз потоку виконання програми,** проводиться для визначення меж можливих шляхів виконання програми, тобто пошуку обмежень поведінки програми. Інформацію про потік виконання може бути внесено програмістом вручну або отримано шляхом автоматизованого аналізу тексту програми. Внаслідок проведення такого аналізу визначаються функції, що викликаються впродовж виконання програми, обмеження ітерацій циклів, залежності між умовними виразами.

Для проведення безпечної оцінки WCET фрагмента програми, інформація про виконання потоку повинна містити дані про всі можливі шляхи виконання програми та максимальну кількість ітерацій циклу.

Наявні підходи для автоматизованого аналізу використовують методи абстрактної інтерпретації, символьного виконання або спеціалізований аналіз потоків даних, що базується на синтаксичному аналізі гілок виконання програми. Вибір того чи іншого підходу залежить від доступних вхідних даних, зокрема: для аналізу тексту програми використовуються підходи [15, 16], для аналізу вихідного або проміжного коду використовуються підходи [17, 18], для аналізу машинного коду застосовуються [19].

Аналіз потоку керування є підґрунтям для подальшого оцінювання тривалості виконання програми. Відтак, вибір методу аналізу є основним завданням при оцінюванні найгіршої тривалості виконання програми.

– Етап 2. Аналіз функцій низького рівня (низькорівневий аналіз), проводиться для визначення можливих меж виконання програмних інструкцій з урахуванням архітектурних особливостей апаратного забезпечення вбудованої системи. Цей вид аналізу потребує доступ до двійкового коду програми, оскільки має аналізувати тривалість виконання кожної інструкції процесором, складність якого описано у розділі 1.

Більшість низькорівневих аналізаторів працюють шляхом створення часової моделі процесора та іншого обладнання, яке впливає на тривалість виконання програмного коду. Розроблені методи припускають, що тривалість виконання команди залежить від даних, які знаходяться в кеш-пам'яті, виконання інструкції відбувається впродовж максимально можливого часу. Окрім цього, методи низькорівневого аналізу мають враховувати модель синхронізації, складність яких залежить від процесора, що використовується. Наприклад, для простих 8 або 16-ти розрядних процесорів модель часової поведінки відносно проста, однак, її створення займає доволі багато часу [20]. Створення моделей для більш складних 32-розрядних процесорів, що містять широкий спектр функціональних блоків, зокрема блоків для збільшення їх продуктивності [21-23], є більш трудомістким і тривалим процесом. Очевидно, що й аналіз таких моделей займає дещо більший час.

Сучасні інструменти низькорівневого аналізу для визначення WCET, зазвичай, повністю автоматизовані. Як параметри аналізу користувач вказує характеристики апаратного забезпечення, зокрема: модель процесора, тактову частоту, обсяг кеш-пам'яті, тип пам'яті. Отримані результати можуть бути покращені шляхом оптимізації процесу роботи з пам'яттю [24].

– Етап 3. Розрахунок найгіршої тривалості виконання, проводиться для об'єднання отриманої інформації про потік виконання програми та часові дані, отримані на попередніх етапах аналізу. У літературі запропоновано три основні види методів розрахунку WCET: на основі дерева програми, шляху виконання та неявного перерахунку шляху. Вибір найбільш доцільного методу розрахунку залежить від базового представлення програми, а також характеристик потоку виконання.

Розрахунок на основі дерева програми проводиться на основі синтаксичного дерева програми [25, 26], що представляє собою орієнтоване дерево, в якому вузли описують структуру програми (послідовності, цикли або умовні об'єкти), а листками дерева представляють основні блоки. Процес розрахунку WCET передбачає трансформацію графу потоку керування (рис. 2a) згідно з правилами, наведеними на рис. 2b.

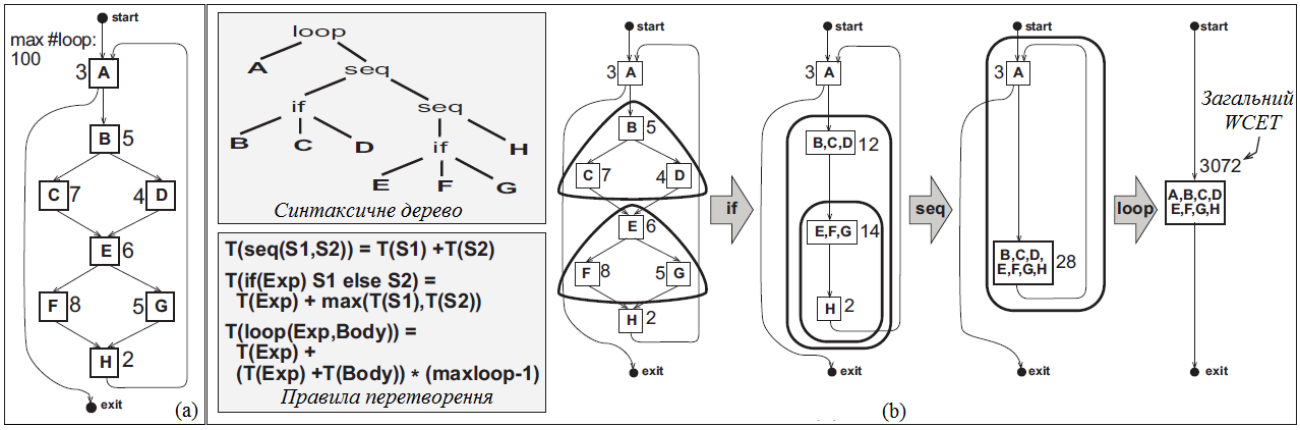

Рис. 2. Процес розрахунку WCET з використанням синтаксичного дерева [15]

Розрахунок на основі шляху виконання, передбачає визначення тривалості виконання фрагментів програми (можливих шляхів виконання), пошуку шляху, що відповідає максимальній тривалості виконання програмного коду, а також поєднання з інформацією про кількість циклів виконання фрагменту програми [27, 28].

Метод неявного перерахунку шляху (Implicit Path Enumeration Technique, IPET) є часто використовуваним методом розрахунку [29-31]. Потік виконання програми та тривалість виконання подають алгебраїчними формулами та/або логічними обмеженнями.

Кожному базовому блоку і ребру графа присвоюється тривалість виконання (t_{entity}) та лічильник виконання (x_{entity}). Розрахунок найгіршої тривалості виконання відбувається шляхом максимізації суми $\sum_{i \in entity} x_i \times t_i$ з урахуванням обмежень, що відображають структуру програми і можливих потоків. В подальшому оцінка WCET проводиться з використанням цілочисельного лінійного програмування (ILP) або шляхом визначення обмежень. Внаслідок цього отримуємо найгіршу кількість виконань кожного вузла чи ребра, а не явний шлях виконання. Тому цей метод зазвичай застосовується для невеликих фрагментів програм.

3. Порівняння методів часового аналізу

В таблиці 1 наведено порівняння існуючих методів часового аналізу, за наступними критеріями: забезпечення повного покриття, незмінність функціоналу та адекватність отриманих результатів. Забезпечення повного покриття програмного коду є основним критерієм вибору методів часового аналізу і класичною вимогою зі стандартів надійності, недотримання якої не дозволить з упевненістю стверджувати про правильність роботи вбудованої системи. Зміна функціоналу є потенційною загрозою достовірності отриманих результатів, незалежно від вибраного методу часового аналізу. Адекватність отриманих результатів відображає похибку методу часового аналізу.

З таблиці 1 видно, що статичні методи часового аналізу забезпечують повне покриття програмного коду, однак завищують отримані результати. Усі динамічні методи часового аналізу забезпечують виконання двох критеріїв, але жоден з них не забезпечує виконання трьох критеріїв.

Таблиця 1

Результати дослідження методів вимірювання тривалості виконання програмного коду

Тип методу	Назва методу	Забезпечує повне покриття	Не змінює функціонал вбудованої системи	Адекватність отриманих результатів
Статичний	Абстрактної інтерпретації	+	-	Завищує
	Символьного виконання	+	-	завищує
	Аналізу потоків даних	+	-	завищує
	Шляху виконання програми	+	-	Завищує
	Неявного перерахунку шляху	+	-	завищує
Динамічний	Вимірювання з допомогою осцилографа та логічного аналізатора	-	-	Не впливає
	Вимірювання з допомогою апаратного трасування	-	-	Не впливає
	Вимірювання з допомогою додаткового програмного забезпечення	-	+	Завищує
	Вимірювання з використанням симуляторів	+	-	Завищує

Висновки

Проведене дослідження методів вимірювання тривалості виконання програмного коду, та їх порівняння за ключовими критеріями показало, що застосування динамічних методів дає змогу:

– відмовитись від розроблення та аналізу моделей поведінки апаратного забезпечення, що

істотно зменшує час, витрачений на аналіз тривалості виконання програмного коду;

– збільшити точність отриманих результатів за рахунок проведення вимірювання на реальному апаратному забезпеченні.

Перспективи подальших досліджень полягають у застосуванні динамічних методів часового аналізу, зокрема, методів, що базуються на вимірюванні тривалості виконання програмного коду з допомогою апаратного трасування.

Створення програмного засобу для часового аналізу з використанням інтерфейсів трасування дасть можливість повністю автоматизувати процес аналізу тривалості виконання програмного коду, отримати повний контроль над вбудованою системою, без внесення змін у її апаратне та програмне забезпечення, що, водночас, не призведе до зміни функціоналу, для виконання якого вбудована система розроблена.

Список використаної літератури

1. Embedded System Market Size, Share, Trends, Report, 2020 [Text]: Report – Radiant Insights Inc., 2015.
2. Stankovic, J. Misconceptions about real-time computing: a serious problem for next-generation systems [Text] / J. Stankovic // Computer. – 1988. – Vol. 21, № 10. – P. 10-19. DOI: 10.1109/2.7053
3. 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
4. Shaw, A. Reasoning about time in higher-level language software [Text] / A. Shaw // IEEE Transactions on Software Engineering. 1989. – Vol. 15, № 7. – P. 875-889. DOI: 10.1109/32.29487
5. Graham, R. Bounds for Certain Multiprocessing Anomalies [Text] / R. Graham // Bell System Technical Journal. 1966. – Vol. 45, № 9. – P. 1563-1581. DOI: 10.1002/j.1538-7305.1966.tb01709.x
6. Reliable and Precise WCET Determination for a Real-Life Processor [Text] / [C. Ferdinand, R. Heckmann, M. Langebach at al.] // Embedded Software. – 2001. – Vol. 2211. – P 469-485. DOI: 10.1007/3-540-45449-7_32
7. MISRA-C 2004 : Guidelines for the use of the C language in critical systems [Text]. Nuneaton: MIRA, 2004.
8. ARM Information Center [Electronic Resource]. – Access Mode : http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.kui0062a/rlarm_ar_rrobin_multit.htm.
9. Stewart, D. Measuring Execution Time and Real-Time Performance [Text] / D. Stewart // In Proceedings of the Embedded Systems Conference (ESCSF), San Francisco, 2004.
10. Zhang, Y. Evaluation of methods for dynamic time analysis for CC systems AB : thesis of Master's degree [Text] / Y. Zhang. – Vasteras: Malardalen University, 2005.
11. CAST: Automating Software Tests for Embedded 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
12. 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, 15-18 October 2012 : proceedings. – Heraklion, ISoLA, 2012. – P. 158-172. DOI: 10.1007/978-3-642-34032-1_17
13. 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
14. 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.
15. Tighten the computation of worst-case execution time by detecting feasible paths [Text] / [H. Aljifri, A. Pons, and M. Tapia] // 19th IEEE International Performance, Computing, and Communications. Conference (IPCCC2000), 5-8 February 2000 : proceedings. – Phoenix, 2000. – P. 23-34. DOI: 10.1109/PCCC.2000.830347
16. Exploiting branch constraints without exhaustive path enumeration [Text] / [T. Chen, T. Mitra, A. Roychoudhury et al.] // 5th International Workshop on Worst-Case Execution Time Analysis (WCET'2005), 5 July 2005 : proceedings. – Palma de Mallorca, 2005. – P. 40-43. DOI: 10.1.1.120.2190
17. Towards a flow analysis for embedded system C programs [Text] / [J. Gustafsson, A. Ermedahl, and B. Lisper] // 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2005), 02 - 04 February 2005: proceedings. – Arizona, 2005. – P. 84-95. DOI: 10.1109/WORDS.2005.53
18. Supporting timing analysis by automatic

bounding of loop iterations [Text] / [C. Healy, M. Sjodin, V. Rustagi et al.] // Real-Time Systems. – 2000. – Vol. 18, № 2-3. – P. 129–156. DOI: 10.1023/A:1008189014032

19. Healy, C. Tighter timing predictions by automatic detection and exploitation of value-dependent constraints [Text] / C. Healy, D. Whalley // 5th IEEE Real-Time Technology and Applications Symposium (RTAS'99), 02 – 04 June 1999 : proceedings. – Vancouver, 1999. – P. 79-92. DOI: 10.1109/RTTAS.1999.777663

20. Engblom, J. Processor Pipelines and Static Worst-Case Execution Time Analysis : PhD thesis [Text] / J. Engblom – Uppsala : Uppsala University, 2002. ISBN 91-554-5228-0.

21. Reliable and precise WCET determination for a real-life processor [Text] / [C. Ferdinand, R. Heckmann, M. et al.] // 1st International Workshop on Embedded Systems, (EMSOFT2000), 8 – 10 October 2001 : proceedings. – London, 2001. – P. 469-485. ISBN: 3-540-42673-6

22. Modeling out-of-order processors for software timing analysis [Text] / [X. Li, A. Roychoudhury, and T. Mitra] // 25th IEEE Real-Time Systems Symposium (RTSS'04), 5 – 8 December 2004 : proceedings. – Libon, 2004. – P. 92-103. DOI: 10.1109/REAL.2004.33

23. Computing the worst case execution time of an avionics program by abstract interpretation [Text] / [J. Souyris, E. Pavec, G. Himbert et al.] // 5th International Workshop on Worst-Case Execution Time Analysis, (WCET'2005) 5 July 2005 : proceedings. – Palma de Mallorca, 2005. – P. 55-59. DOI: 10.4230/OASIcs.WCET.2005.810

24. Convenient user annotations for a WCET tool [Text] / [C. Ferdinand, R. Heckmann, and H. Theiling] // 3rd International Workshop on Worst-Case Execution Time Analysis, (WCET'2003), 1 July 2003 : proceedings. – Porto, 2003. – P. 17–20.

25. An accurate worst-case timing analysis for RISC processors [Text] / [S. Lim, Y. Bae, C. Jang et al.] // IEEE Transactions on Software Engineering. – 1995. – Vol. 21, № 7. – P. 593–604. DOI: 10.1109/32.392980

26. Park, C. Experiments with a program timing tool based on a source-level timing schema [Text] / C. Park, A. Shaw // 11th IEEE Real-Time Systems Symposium (RTSS'90), 5-7 December 1990 : proceedings. – Lake Buena Vista, 1990. – P. 72-81. DOI: 10.1109/REAL.1990.128731

27. Bounding pipeline and instruction cache Performance [Text] / [C. Healy, R. Arnold, F. Muller et al.] // IEEE Transactions on Computers. – 1999. – Vol. 48, № 1. – P. 53–70. DOI: 10.1109/12.743411

28. Peter, F. Complete worst-case execution time analysis of straight-line hard real-time programs [Text] / F. Peter, S. Altenbernd // Journal of Systems Architecture. – 1999. – Vol. 46, № 4. – P. 339–355. DOI: 10.1016/S1383-7621(99)00010-7

29. Worst-case execution-time analysis for digital signal processors [Text] / [N. Holsti, T. Langbacka, and S. Saarinen] // X European Signal Processing Conference (EUSIPCO 2000), 4-8 September 2000 : proceedings. – Tampere, 2000.

30. Li, Y-T. Performance analysis of embedded software using implicit path enumeration [Text] / Y-T. S. Li, S. Malik // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 1997. – Vol. 16, № 12. – P. 1477 - 1487. DOI: 10.1109/43.664229

31. Puschner, P. Computing maximum task execution times – a graph-based approach [Text] / P. Puschner, A. Schedl // Journal of Real-Time Systems – 1997. – Vol. 13, № 1. – P. 67 - 91. DOI: 10.1023/A:100790500

References

1. Embedded System Market Size, Share, Trends, Report, 2020 : Report – Radiant Insights Inc., 2015.

2. J. Stankovic, (1988) "Misconceptions about real-time computing: a serious problem for next-generation systems",Computer, vol. 21, no. 10, pp. 10-19, DOI: 10.1109/2.7053

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

4. A. Shaw, (1989) "Reasoning about time in higher level language software", IEEE Transactions on Software Engineering, vol. 15, no. 7, pp. 875-889, DOI: 10.1109/32.29487

5. R. Graham, (1966) "Bounds for Certain Multipr-cessing Anomalies", Bell System Technical Journal, vol. 45, no. 9, pp. 1563-1581,. DOI: 10.1002/j.1538-7305.1966.tb01709.x

6. C. Ferdinand, R. Heckmann, M. Langebach, F. Martin, M. Schmidt, H. Theiling, S. Thesing and R. Wilhelm, (2001) "Reliable and Precise WCET Determination for a Real-Life Processor", Embedded Software, pp. 469-485, DOI: 10.1007/3-540-45449-7_32

7. MISRA-C 2004 : Guidelines for the use of the C language in critical systems. Nuneaton: MIRA, 2004.

8. ARM Information Center [Electronic Resource]. – Access Mode : http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.kui0062a/rlarm_ar_rrobin_multit.htm.

9. D. Stewart, (2004) "Measuring Execution Time and Real-Time Performance", In Proceedings

of the Embedded Systems Conference (ESCSF), San Francisco.

10. Y. Zhang, (2005) "Evaluation of methods for dynamic time analysis for CC systems AB : thesis of Master's degree", Vasteras: Malardalen University.

11. 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

12. 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

13. D. Fedasyuk, R. Chopey and B. Knysh, (2017) "Ar-chitecture 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), pp. 278-282, DOI: 10.1109/CADSM.2017.7916134

14. 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).

15. H. Aljifri, A. Pons and M. Tapia, (2000) "Tighten the computation of worst-case execution-time by detecting feasible paths", Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference (Cat. No.00CH37086). DOI: 10.1109/PCCC.2000.830347

16. T. Chen, T. Mitra, A. Roychoudhury, (2005), "Exploiting branch constraints without exhaustive path enumeration", 5th International Workshop on Worst-Case Execution Time Analysis (WCET'2005), pp. 40-43. DOI: 10.1.1.120.2190

17. J. Gustafsson, A. Ermedahl and B. Lisper, "Towards a Flow Analysis for Embedded System C Programs", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. DOI: 10.1109/WORDS.2005.53

18. C. Healy, M. Sjödin, V. Rustagi, D. Whalley and R. Engelen, Real-Time Systems, vol. 18, no. 23, pp. 129-156, 2000. DOI: 10.1023/A:1008189014032

19. C. Healy and D. Whaley, "Tighter timing predictions by automatic detection and exploitation of value-dependent constraints", Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium. DOI: 10.1109/RTTAS.1999.777663

20. J. Engblom, (2002) Processor pipelines and static worst-case execution time analysis. Uppsala: Acta Universitatis Upsaliensis,. ISBN 91-554-5228-0.

21. C. Ferdinand, R. Heckmann, (2001), "Reliable and precise WCET determination for a real-life processor", 1st International Workshop on Embedded Systems, (EMSOFT2000).

22. Xianfeng Li, A. Roychoudhury and T. Mitra, (2004), "Modeling Out-of-Order Processors for Software Timing Analysis", 25th IEEE International Real-Time Systems Symposium. DOI: 10.1109/REAL.2004.33

23. J. Souyris, E. Pavec, G. Himbert, (2005), "Compu-ting the worst case execution time of an avionics program by abstract interpretation", 5th International Workshop on Worst-Case Execution Time Analysis, (WCET'2005). DOI: 10.4230/OASIcs.WCET.2005.810

24. C. Ferdinand, R. Heckmann, and H. Theiling, (2003), "Convenient user annotations for a WCET tool", 3rd International Workshop on Worst-Case Execution Time Analysis, (WCET'2003).

25. Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon and Chong Sang Kim, (1995), "An accurate worst case timing analysis for RISC processors", IEEE Transactions on Software Engineering, vol. 21, no. 7, pp. 593-604. DOI: 10.1109/32.392980

26. C. Park and A. Shaw, (1990) "Experiments with a program timing tool based on source-level timing schema", Proceedings 11th Real-Time Systems Symposium, DOI: 10.1109/REAL.1990.128731

27. C. Healy, R. Arnold, F. Mueller, D. Whalley and M. Harmon, (1999), "Bounding pipeline and instruction cache performance", IEEE Transactions on Computers, vol. 48, no. 1, pp. 53-70. DOI: 10.1109/12.743411

28. F. Stappert and P. Altenbernd, (2000), "Complete worst-case execution time analysis of straight-line hard real-time programs", Journal of Systems Architecture, vol. 46, no. 4, pp. 339-355. DOI: 10.1016/S1383-7621(99)00010-7

29. N. Holsti, T. Langbacka, and S. Saarinen, (2000),"Worst-case execution-time analysis for digital signal processors", X European Signal Processing Conference (EUSIPCO 2000), 4-8 September .

30. Y-T. S. Li, S. Malik, (1997),"Performance analysis of embedded software using implicit path enumeration", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 12, pp. 1477-1487. DOI: 10.1109/43.664229

31. P. Puschner, A. Schedl, (1997), "Computing maxi-mum task execution times – a graph-based approach", Journal of Real-Time Systems, vol. 13, no 1, pp. 67-91. DOI: 10.1023/A:100790500.

OVERVIEW OF APPROACHES TO THE EXECUTION TIME ANALYSIS OF PROGRAMS

R. S. Chopey, D. V. Fedasyuk

Lviv Polytechnic National University

Abstract. *We've considered common problems that typically occur when analyzing the execution time of software in general and determined the problems of analyzing the execution time of microcontroller-based embedded systems, particularly. The authors have studied the existing methods of static and dynamic software execution time analysis.*

Upon the obtained research results one may conclude that using dynamic methods for execution time analysis would allow us to:

– eliminate development and analysis of hardware models that will reduce significantly the duration required for software execution time;

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

Upon the research we can state that there exists a lack of developing a software for execution time analysis using trace interfaces. This would allow us to completely control an embedded system and the process of testing its execution time without any amendments in its hardware and software.

Keywords: *execution time analysis, real-time embedded systems, CMSIS-RTOS RTX, worst-case execution time, WCET, best-case execution time, BCET.*

ОБЗОР ПОДХОДОВ К АНАЛИЗУ ПРОДОЛЖИТЕЛЬНОСТИ ВЫПОЛНЕНИЯ ПРОГРАММНОГО КОДА

Р. С. Чопей, Д. В. Федасюк

Национальный университет «Львовская политехника»

Аннотация. *Рассмотрены общие проблемы, возникающие при анализе продолжительности выполнения программного кода, определены актуальные проблемы для микроконтроллерных встроенных систем. Осуществлен обзор методов статического и динамического анализа продолжительности выполнения программного кода. Доказана необходимость разработки программного обеспечения для временного анализа с использованием интерфейсов трассировки, что позволит полностью контролировать встроенную систему и процесс тестирования продолжительности выполнения программного кода без внесения изменений в ее программное и аппаратное обеспечение.*

Ключевые слова: *анализ продолжительности выполнения, встроенные системы реального времени, CMSIS-RTOS RTX, наихудшая продолжительность выполнения, наилучшая продолжительность выполнения.*

Отримано 09.09.2017

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

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

ORCID ID: 0000-0003-3552-7454