МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 08.00.00.000 ПЗ

Група ІПм-23-1

Бойчук Віталій

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Бойчук Віталій Ігорович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методології генерації коду для багатоядерних архітектур

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Бойчук В.І.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Храбатин Роман Ігорович, к.ф.-м.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Бойчуку Віталію Ігоровичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методології генерації коду для багатоядерних архітектур”

керівник проекту (роботи) Храбатин Роман Ігорович, к.ф.-м.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій генерації коду

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження області розробки програмного забезпечення для багатоядерних архітектур
2. Особливості розробки методології для багатоядерних архітектур
3. Методи, принципи та підходи до генерації коду для багатоядерних систем
4. Імплементація методів та методології генерації коду для багатоядерних архітектур

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Архітектура AUTOSAR (рис. 1.1)
2. Схема підтримки Simulink стандарту AUTOSAR (рис. 1.2)
3. Принцип роботи інструментарію Motar (рис. 1.7)
4. Графічний огляд вимог проекту (рис. 1.10)
5. Візуалізація операційного середовища системи (рис. 1.12)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження області розробки програмного забезпечення для багатоядерних архітектур	15.10.2024	виконано
4	Особливості розробки методології для багатоядерних архітектур	08.11.2024	виконано
5	Методи, принципи та підходи до генерації коду для багатоядерних систем	20.11.2024	виконано
6	Імплементація методів та методології генерації коду для багатоядерних архітектур	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 75 с., 33 рис., 6 табл., 50 джерел.

Тема: Методології генерації коду для багатоядерних архітектур

Об'єкт дослідження: процес розробки програмного забезпечення для систем на базі багатоядерних архітектур, сумісних з вимогами функціональної безпеки.

Мета роботи: розробка методології генерації коду для багатоядерних архітектур з урахуванням вимог стандартів та аналізу можливостей поділу задач для багатоядерної обробки.

Предмет дослідження: методи та методології генерації коду для багатоядерних архітектур із використанням інструментарію Motar Multi-Core Extension.

Результати дослідження:

В роботі розроблено методологію генерації коду для багатоядерних архітектур на основі інструментарію Motar Multi-Core Extension, що забезпечує підвищену продуктивність і відповідність стандартам функціональної безпеки.

Висновок

Розроблені в дослідженні методології можуть бути використані для вдосконалення процесів розробки програмного забезпечення для блоків керування з багатоядерними архітектурами, що підвищить їх продуктивність і відповідність вимогам функціональної безпеки.

БАГАТОЯДЕРНІ АРХІТЕКТУРИ, ГЕНЕРАЦІЯ КОДУ, ФУНКЦІОНАЛЬНА БЕЗПЕКА, СИСТЕМНИЙ АНАЛІЗ, ПОДІЛ ЗАДАЧ, БЛОКИ КЕРУВАННЯ, ФУНКЦІОНАЛЬНА БЕЗПЕКА

ABSTRACT

Master Thesis: 75 pp., 33 fig., 6 tab., 50 sources.

Thesis Subject: Topic: Code generation methodologies for multi-core architectures

The object of research: the process of developing software for systems based on multicore architectures, compatible with the requirements of functional safety.

The purpose of the work: development of a code generation methodology for multi-core architectures taking into account the requirements of standards and analysis of the possibilities of dividing tasks for multi-core processing.

Research subject: methods and methodologies of code generation for multi-core architectures using the Motar Multi-Core Extension toolkit.

Research results

The work developed a code generation methodology for multi-core architectures based on the Motar Multi-Core Extension toolkit, which ensures increased performance and compliance with functional safety standards.

Conclusion

The methodologies developed in the research can be used to improve software development processes for control units with multi-core architectures, which will increase their performance and compliance with functional safety requirements.

MULTI-CORE ARCHITECTURES, CODE GENERATION, FUNCTIONAL SAFETY, SYSTEM ANALYSIS, DIVISION OF TASK, CONTROL UNITS, FUNCTIONAL SAFETY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ASIL - Automotive Safety Integrity Level

AUTOSAR - Automotive Open System Architecture

BSW - Basic Software

CAN - Controller Area Network

CPU - Central Processing Unit

ECU - Electronic Control Unit

HARA- Hazard Analysis and Risk Assessment

I/O - Input and Output

LED - Light-Emitting Diode

LIN - Local Interconnect Network

MIL - Model-In-the-Loop

MMCE - Motar Multi-Core Extension

MPUC - Model Partitioning Use Case

OS - Operating System

RTE - Runtime Environment

TRIZ - Theory of Inventive Problem-Solving

UDP - User Datagram Protocol

ВСТУП

Актуальність теми.

Розробка програмного забезпечення для багатоядерних архітектур є складною задачею, що вимагає глибокого розуміння як апаратних особливостей процесорів, так і принципів паралельного програмування. Аналіз існуючих інструментальних засобів показав, що незважаючи на різноманітність пропонованих рішень, жоден з них не забезпечує повного набору функціональних можливостей для ефективної розробки програмного забезпечення для широкого спектра багатоядерних платформ. Особливістю багатоядерних систем є необхідність враховувати такі фактори, як кеш-пам'ять, комунікації між ядрами, а також специфіку операційної системи. На основі проведеного аналізу сформульовані наступні вимоги до розроблюваної методології: підтримка різних моделлю паралелізму, автоматизація процесу розпаралелювання, забезпечення портативності програмного забезпечення та ін.

Для скорочення часу розробки програмного забезпечення для автомобільних застосувань існує платформа Motar. Платформа Motar є інструментарієм на базі Simulink, який допомагає неспеціалістам у галузі програмного забезпечення розробляти програмне забезпечення. Нещодавно стала доступною електронна контрольна одиниця (ECU) з багатоядерним процесором як цільове обладнання для генерації коду. Це дослідження пропонує рішення для генерації коду для такої багатоядерної цільової платформи, яке відповідає стандарту функціональної безпеки.

Протягом цього дослідження були вивчені архітектурні шаблони, які в даний час використовуються в автомобільній промисловості для критично важливих для безпеки застосувань. Під час дослідження цих архітектурних шаблонів концепція E-Gas була обрана як оптимальна основа для розширення Motar Multi-Core (MMCE). Був проведений системний аналіз для визначення високорівневих вимог, випадків використання, системної

архітектури та поведінки системи для MMCE. Для одного з визначених випадків використання, випадку використання розподілу моделі (MPUC), були визначені необхідні кроки розробки.

У сучасних автомобільних системах зростає попит на високоефективні, безпечні та надійні рішення, здатні забезпечити продуктивність на рівні багатоядерних архітектур, зокрема в контексті стандартів функціональної безпеки, таких як ISO 26262. Створення методологій генерації коду для таких архітектур є важливим завданням, що дозволяє інтегрувати сучасні технології в автомобільні електронні блоки керування (ECU). Інструмент Motar Multi-Core Extension (MMCE) надає нові можливості для розробки багатоядерних систем, але вимагає дослідження й оптимізації для конкретних випадків використання. Актуальність дослідження полягає в необхідності створення ефективних методів генерації коду для багатоядерних архітектур з підтримкою функціональної безпеки.

Мета дослідження - розробка методології генерації коду для багатоядерних архітектур з урахуванням вимог стандартів та аналізу можливостей поділу задач для багатоядерної обробки.

Об’єкт дослідження - процес розробки програмного забезпечення для систем на базі багатоядерних архітектур, сумісних з вимогами функціональної безпеки

Предмет дослідження - методи та методології генерації коду для багатоядерних архітектур із використанням інструментарію Motar Multi-Core Extension (MMCE).

Відповідно до мети роботи було сформовано наступні **задачі:**

- Провести огляд існуючих методологій генерації коду для багатоядерних архітектур в автомобільній індустрії.
- Визначити вимоги до інструментарію Motar Multi-Core Extension для розробки програмного забезпечення багатоядерних систем.

- Дослідити можливості поділу задач та спільного використання ресурсів у рамках багатоядерних системах.
- Проаналізувати та оптимізувати варіанти використання.
- Розробити технічне рішення для генерації коду, сумісного зі стандартом.
- Оцінити функціональну безпеку та продуктивність розробленого технічного рішення.

Методи дослідження.

- Огляд та аналіз наукової літератури з теми багатоядерних архітектур і інструментаріїв для генерації коду.

- Системний аналіз інструментарію Motar Multi-Core Extension.

- Моделювання варіантів використання багатоядерної архітектури в середовищі Simulink.

- Технічний аналіз та симуляція поділу задач з використанням Model Partitioning Use Case (MPUC).

Наукова новизна отриманих результатів.

Розроблено методологію генерації коду для багатоядерних архітектур на основі інструментарію Motar Multi-Core Extension, що забезпечує підвищену продуктивність і відповідність стандартам функціональної безпеки. Визначено нові функціональні можливості для оптимізації розподілу завдань між ядрами багатоядерного процесора та спільного використання ресурсів.

Практичне значення магістерської роботи.

Розроблені в дослідженні методології можуть бути використані для вдосконалення процесів розробки програмного забезпечення для блоків керування з багатоядерними архітектурами, що підвищить їх продуктивність і відповідність вимогам функціональної безпеки. Запропоновані рішення

також можуть бути інтегровані в інструментарій Motar для підтримки стандартів, що забезпечить інструменти для ефективної розробки безпечних та надійних систем.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 75 сторінок, і містить 33 рисунки, 6 таблиць, список використаних джерел із 50 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ БАГАТОЯДЕРНИХ АРХІТЕКТУР

1.1. Опис проблеми дослідження

Даний розділ містить контекст для проблеми, яка є предметом цього дослідження. Крім того, представлено контекст проекту та надано опис структури роботи.

Протягом останніх років група ICT розробила платформу, щоб допомогти інженерам, які не займаються програмним забезпеченням, створювати програмне забезпечення. Ця платформа під назвою Motar дозволяє розширювати модель Simulink таким чином, щоб генерувався вихідний код на основі AUTOSAR. Цей згенерований код потім компілюється та завантажується на сумісний електронний блок керування (ECU). Із зростанням клієнтської бази платформи Motar зростає попит на сертифікацію платформи Motar за стандартом ISO 26262. Таким чином, майбутні кроки для сертифікації Motar були визначені під час попередніх досліджень. Базуючись на висновках у [1], для цього дослідження передбачається, що інструмент Motar уже сертифікований. Проте можливість сертифікації платформи Motar не гарантує, що розроблена програма також сертифікована за стандартом ISO. Щоб отримати сертифікат ISO, уся програма має бути розроблена відповідно до стандарту. Це дослідження має на меті розширити поточну версію Motar, щоб клієнт мав більше варіантів дизайну для задоволення вимог безпеки. У поточному випуску Motar код генерується для ЕБУ, що містить багатоядерний процесор. Однак потенціал цього багатоядерного процесора наразі не використовується повною мірою, оскільки повний вихідний код програми працює на одному ядрі процесора, як і в одноядерному процесорі. Щоб надати клієнтам більше варіантів дизайну для свого програмного забезпечення, це дослідження зосереджено на

використанні багатоядерного процесора. Оскільки попит на функціонально безпечну розробку зростає, здатність розробити функціонально безпечну програму є головною увагою.

З переходом на новий багатоядерний ECU група ICT вирішила, що Motar повинна мати можливість генерувати код для багатоядерних архітектур. Наразі невідомо, яке розширення чи модифікація платформи Motar необхідна для досягнення цієї мети. Тому це дослідження зосереджено на розробці та реалізації генерації коду Motar для багатоядерних архітектур. Однак, крім проблеми генерації коду, також зростає попит на розробку додатків, сумісних із ISO 26262. Таким чином, розширення генерації багатоядерного коду має мати можливість генерувати код для багатоядерної архітектури, сумісної зі стандартом ISO.

Оскільки Motar створює вихідний код на основі AUTOSAR, у цьому розділі наведено вступ до AUTOSAR, як представлено в [1]. AUTOSAR — це стандартизована архітектура програмного забезпечення для застосування в автомобільному секторі, яка складається з чотирьох рівнів [2], як показано на рисунку 1.1.

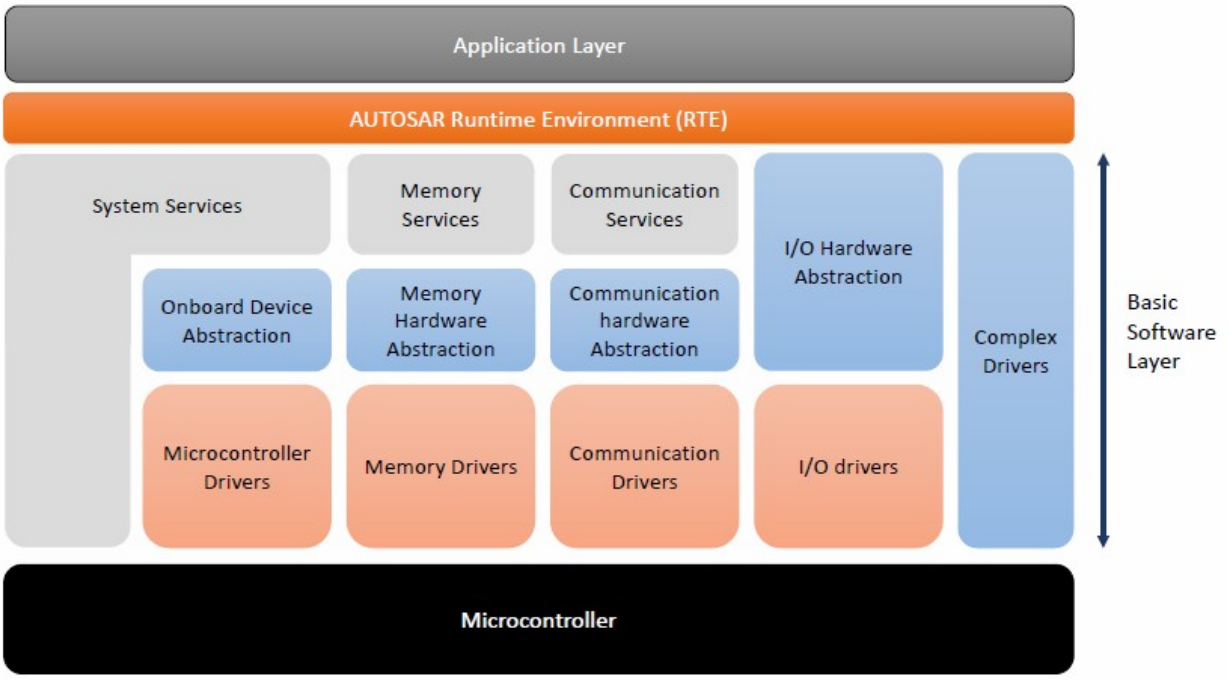

Рис. 1.1. Архітектура AUTOSAR [1]

Верхній рівень, який є прикладним рівнем, містить програмні компоненти, що стосуються конкретної програми. Ці конкретні компоненти програмного забезпечення можуть, наприклад, бути компонентами програмного забезпечення для приводів і датчиків. Другий рівень, рівень AUTOSAR Runtime Environment (RTE), забезпечує комунікаційні послуги для прикладного програмного забезпечення. Через цей рівень різні програмні компоненти AUTOSAR можуть спілкуватися з іншими компонентами. Цей зв'язок може використовуватися як між блоками управління, так і всередині блоків керування. Третій рівень — базове програмне забезпечення (BSW). На рисунку 1.1 показано, що BSW має кілька компонентів.

Як видно, можна виділити чотири основні стеки програмного забезпечення:

- Системний стек: системний стек складається з трьох компонентів: системних служб, абстракції бортового пристрою та драйверів мікроконтролера. У цьому стеку сервіси не тільки знаходяться на вершині абстракції, але й з’єднують RTE та мікроконтролер. Це тому, що системні служби також включають операційну систему. Основне завдання цього стеку полягає в обробці помилок, звітності та діагностиці.

- Стек пам’яті: Стек пам’яті складається з компонента служби, апаратної абстракції та драйвера. Завдання цих компонентів — забезпечити механізми доступу до внутрішніх і зовнішніх пристроїв енергонезалежної пам’яті.

- Комунікаційний стек: комунікаційний стек складається з сервісу, апаратної абстракції та компонента драйверів, як і попередні два стеки. Комунікаційний стек забезпечує інтерфейс для кількох протоколів зв’язку, таких як мережа контролера (CAN), локальна мережа з’єднання (LIN) і Ethernet.

- Стек вводу-виводу (I/O): Стек вводу-виводу забезпечує інтерфейс сигналу вводу-виводу. На відміну від розглянутих раніше стеків, стек

вводу/виводу складається лише з апаратної абстракції та компонента драйвера.

Окрім цих чотирьох стеків, є також складний компонент драйверів. Цей складний компонент драйверів використовується для реалізації нестандартизованої функціональності, але виходить за межі цього дослідження. Однак чотири стеки утворюють основу для платформи Motar. Слід зазначити, що різні компоненти мають різні характеристики самі по собі, а також між стеками кілька компонентів працюють разом. Більше інформації про цю тему доступно в [2].

Основні характеристики AUTOSAR:

- Модульність: Архітектура AUTOSAR дозволяє розробникам створювати програми з окремих модулів, які легко можна адаптувати чи заміняти без впливу на інші компоненти системи. Це забезпечує високу гнучкість під час розробки.

- Масштабованість: AUTOSAR підтримує різні типи автомобілів, від малих машин до великих комерційних транспортних засобів. Це досягається через стандартизовані інтерфейси та гнучку архітектуру.

- Сумісність: Стандартизовані програмні інтерфейси дозволяють виробникам автомобілів інтегрувати компоненти від різних постачальників без значних змін у системах. Це спрощує процес розробки та знижує витрати.

- Реалізація розподілених систем: AUTOSAR підтримує розподілені системи, що працюють на різних електронних блоках керування (ECU) в автомобілі. Це важливо для забезпечення координації між різними компонентами автомобіля, такими як системи безпеки, комфорту та допомоги водію.

- Розділення апаратної та програмної частини: AUTOSAR розділяє функціональну логіку від апаратної платформи, що дозволяє використовувати різні ECU, забезпечуючи незалежність програмних рішень від конкретної апаратної архітектури.

Архітектура AUTOSAR складається з кількох шарів:

- Basic Software (BSW): Це базове програмне забезпечення, яке включає драйвери для апаратних компонентів, системні сервіси, а також засоби комунікації та діагностики. BSW функціонує як проміжне програмне забезпечення, яке абстрагує апаратне забезпечення від додатків.

- Runtime Environment (RTE): Цей шар виконує функції комунікації між додатками (Application Layer) та базовим програмним забезпеченням (BSW). RTE гарантує, що всі компоненти можуть взаємодіяти без залежності від фізичного розташування або апаратних обмежень.

- Application Layer: Тут знаходяться всі програмні компоненти, які виконують конкретні функції автомобіля, такі як контроль гальм, керування двигуном, системи комфорту, мультимедіа тощо.

Переваги AUTOSAR:

- Підтримка повторного використання коду: Модульний підхід та стандартизація інтерфейсів дозволяють використовувати один і той самий код для різних проектів або транспортних засобів.

- Покращена якість і безпека: Завдяки стандартизованому процесу розробки можна досягти високої надійності та безпеки програмного забезпечення, яке відповідає суворим вимогам автомобільного сектору.

- Спрощення інтеграції: Постачальники можуть розробляти свої модулі окремо, що полегшує інтеграцію різних компонентів у складну автомобільну систему.

Інструментальне середовище Simulink® надає вбудовану підтримку стандарту AUTOSAR. Для розробки програмного забезпечення AUTOSAR у Simulink використовують такі можливості:

- Проектування та симуляція класичних та адаптивних систем AUTOSAR: за допомогою Simulink та набору блоків AUTOSAR Blockset.

- Створення ієрархічних композицій та компонентів програмного забезпечення AUTOSAR: за допомогою інструменту System Composer™.

- Генерація описів ARXML та виробничого коду C або C++: за допомогою Embedded Coder® для подальшого тестування та інтеграції з середовищем виконання AUTOSAR (AUTOSAR RTE).

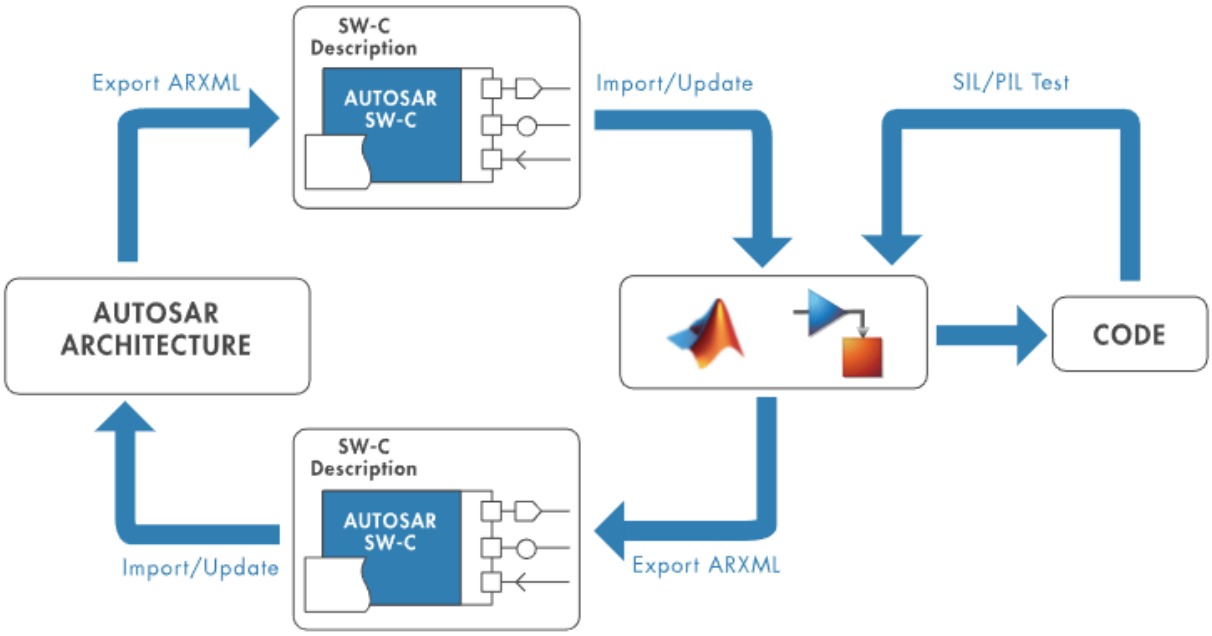

Рис. 1.2. Схема підтримки Simulink стандарту AUTOSAR

Simulink, AUTOSAR Blockset та Embedded Coder забезпечують двонаправлену інтеграцію з архітектурами AUTOSAR, що дозволяє:

- Створення та експорт файлів ARXML: за допомогою інструменту створення AUTOSAR або Simulink. Ці файли використовуються для розробки програмних компонентів AUTOSAR (SWC).

- Імпорт SWC до Simulink: для генерації або оновлення моделей, а також генерації коду C та проведення симуляцій у режимі моделі в петлі (SIL) або процесора в петлі (PIL).

- Експорт оновлених файлів ARXML: для синхронізації з іншими інструментами розробки AUTOSAR.

AUTOSAR забезпечує спільну платформу для розвитку інноваційних автомобільних систем, що підтримує тенденції автоматизації, підключення до Інтернету та електрифікації в автомобілях.

1.2. Дослідження інструментарію розробки програмного забезпечення для багатоядерних архітектур

Motar — це інструментарій, який розроблено для інтеграції платформи Simulink з архітектурою AUTOSAR. Він спрощує процес створення, тестування та впровадження програмного забезпечення для автомобільних електронних блоків керування (ECU), використовуючи інструменти моделювання та симуляції Simulink. Motar автоматизує значну частину роботи, пов'язану з генерацією коду, сумісного з AUTOSAR, що робить його зручним рішенням для інженерів, які працюють у автомобільній галузі.

Motar надає можливість розробляти моделі програмного забезпечення безпосередньо в Simulink, автоматично перетворюючи їх на код, що сумісний з AUTOSAR. Підтримка модульного підходу дозволяє розробникам легко адаптувати або змінювати моделі без впливу на всю систему.

Motar автоматично генерує AUTOSAR-сумісний код на основі моделей Simulink. Це включає як реалізацію функціональної логіки, так і опис необхідних інтерфейсів, сервісів і комунікаційних механізмів. Використання стандартів AUTOSAR гарантує, що розроблені системи будуть легко інтегровані з іншими компонентами автомобіля. Motar дозволяє використовувати Simulink для моделювання динаміки роботи систем і компонентів автомобіля, включаючи електроніку та механіку. Це дає можливість тестувати функціональність системи ще на ранніх стадіях розробки, перед безпосередньою інтеграцією у фізичне середовище.

Motar автоматизує процеси генерації коду та його інтеграції з платформою AUTOSAR, що значно скорочує час на розробку та знижує можливість виникнення помилок. Вбудовані інструменти дозволяють автоматично створювати конфігурації для ECU, що відповідають специфікаціям AUTOSAR, з мінімальною участю розробника.

Motar добре підходить для роботи над великими проектами, де використовуються складні архітектури з багатьма ECU. Він дозволяє легко масштабувати розроблені системи для різних типів автомобілів та платформ. Використання Simulink у поєднанні з Motar дає можливість створювати прототипи систем ще на ранніх етапах розробки. Це дозволяє інженерам швидко перевіряти нові ідеї та вдосконалювати їх.

Процес роботи з Motar:

- Моделювання в Simulink: Розробник створює модель системи в Simulink, яка описує функціональну логіку та взаємодію між компонентами.

- Генерація AUTOSAR-сумісного коду: За допомогою Motar модель автоматично перетворюється на AUTOSAR-сумісний код з усіма необхідними інтерфейсами та конфігураціями.

- Інтеграція з ECU: Згенерований код завантажується на цільові автомобільні ECU для тестування та використання в реальних умовах.

- Тестування та відлагодження: Завдяки інтеграції з Simulink, тестування моделей можна виконувати безпосередньо в симуляційному середовищі, до того як код буде впроваджено на фізичні ECU.

Переваги Motar:

- Зменшення складності розробки: Інструментарій значно спрощує процес інтеграції програмного забезпечення з архітектурою AUTOSAR, що дозволяє зосередитися на функціональних аспектах, а не на технічних деталях конфігурації.

- Підвищення ефективності: Автоматизація генерації коду та використання готових компонентів допомагають скоротити час на розробку та тестування.

- Зниження ризику помилок: Завдяки автоматизації конфігурації та генерації коду, Motar допомагає уникнути багатьох типових помилок, які можуть виникнути при ручній роботі.

Щоб представити платформу Motar, у цьому розділі обговорюється основний принцип роботи платформи, як представлено в [1]. Для початку

Motar — це інструментарій для Simulink, який складається з кількох блоків, розділених на чотири категорії. Ці чотири категорії базуються на рівні AUTOSAR BSW і тому називаються SysStack, MemStack, ComStack і IoStack. Усі ці категорії відповідають чотирьом категоріям, згаданим у вступі до AUTOSAR.

1.2.1. Основні блоки інструментарію

IoStack – категорія яка представляє вхідні та вихідні сигнали цільової електронної контрольної одиниці (ECU). Як видно на рисунку 1.3, категорія складається з шести різних блоків. Функціональність кожного блоку описана в [25] і буде коротко описана нижче.

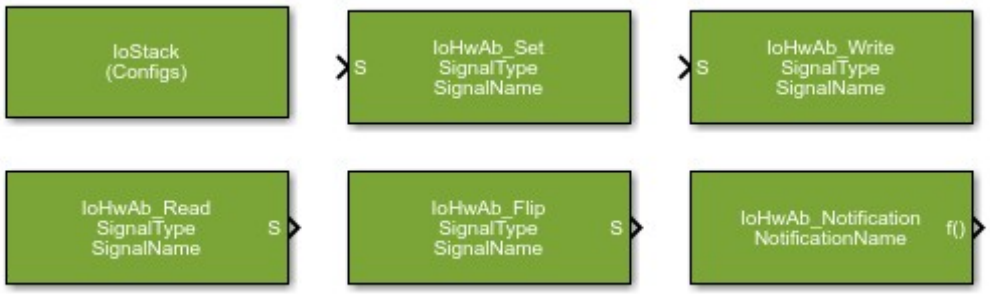

Рис. 1.3. Огляд блоків IoStack

Перший блок - це блок IoStack Configs. Під час роботи з Motar цей блок завжди повинен бути присутнім у моделі для компіляції моделі. У графічному інтерфейсі користувача (GUI) цього блоку можна налаштувати драйвери для різних вхідних та вихідних сигналів. Кожен сигнал, що використовується іншими блоками IoStack, спочатку повинен бути налаштований у цьому GUI. Перший блок, який вимагає налаштованих сигналів, - це блок IoHwAb Write. Цей блок можна використовувати для запису значення на налаштований цифровий вихідний канал. Щоб встановити значення на налаштований аналоговий вихідний канал, можна використовувати блок IoHwAb Set. Крім запису значень на порт, також

можна зчитувати вхід на порт. Це можна зробити за допомогою блоку IoHwAb Read, який повертає поточне значення на вибраному каналі. Крім запису та читання значень, також є блок, який може створити тригер. Це блок IoHwAb Flip, який генерує тригер перемикання щоразу, коли значення з вибраного вхідно-вихідного каналу змінюється. На додаток до цього, як блок IoHwAb Flip, так і блок IoHwAb Read можуть генерувати вхідний сигнал симуляції. Це сигнал, який використовується лише під час симуляції в середовищі Simulink і підтримує тестування Model-In-the-Loop (MIL).

Наступний рівень - це ComStack. Ці блоки представляють комунікаційні сигнали цільового ECU. На рисунку 1.4 показані блоки з цієї категорії.

Рис. 1.4. Огляд блоків ComStack

Основним блоком ComStack є блок ComStack Configs. Як і в IoStack, цей блок завжди повинен бути присутнім у моделі Simulink для компіляції. У графічному інтерфейсі користувача (GUI) цього блоку можна налаштувати всі драйвери та послуги комунікації. Всередині блоку конфігурації встановлюються повідомлення CAN (Controller Area Network) та UDP (User Datagram Protocol). Щоб відправити повідомлення CAN або UDP, використовується блок Com SendSignal. Цей блок відправить вхідне значення сигналу S від цілі через комунікаційний рівень. Щоб отримати повідомлення, слід використовувати блок Com ReceiveSignal. Крім значень сигналу S, цей блок також виведе таймаут T. Цей таймаут вказує, чи не отримано сигнал протягом налаштованого таймауту.

Третій стек - це SysStack. Цей стек використовується для кількох специфічних для ECU функцій та управління ECU. Рисунок 1.5 показує, які блоки присутні в наборі інструментів.

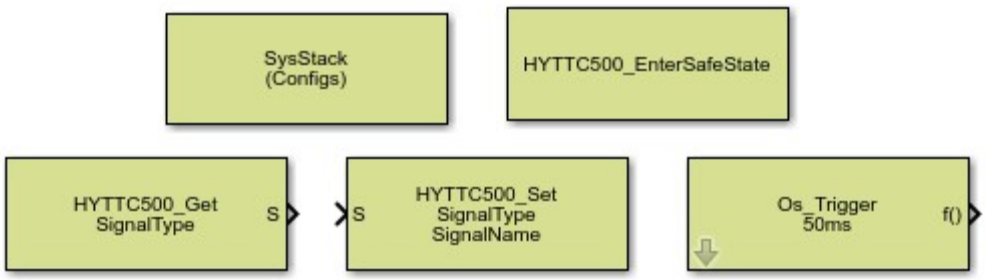

Рис. 1.5. Огляд блоків SysStack

Як і в попередніх двох стеках, SysStack також має блок конфігурації. Цей блок називається SysStack Configs. Цей блок повинен бути включений у модель Simulink для компіляції. Через GUI цього блоку можна налаштувати драйвери для системного стека. Коли драйвери налаштовані, інші блоки SysStack надають функціональності. Перший блок, HYTTC500 EnterSafeState, використовується для запуску безпечного стану цільового ECU. Під час використання цього блоку в Simulink його необхідно використовувати в підсистемі з тригером, і після запуску немає повернення до нормальної роботи через модель. Щоб моніторити ECU, використовується блок HYTTC500 Get. Остання функціональність, що пропонується SysStack, - це Os Trigger.

Остаточний стек - це MemStack, який використовується для функцій пам'яті. Рисунок 1.6 відображає різні блоки Simulink.

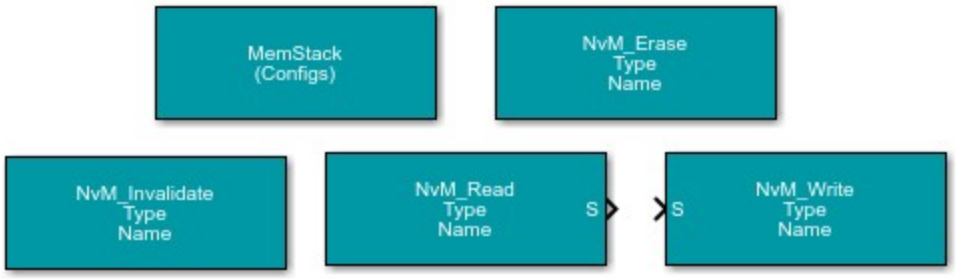

Рис. 1.6. Огляд блоків MemStack

У кінцевому стеку також присутній блок конфігурації. У GUI MemStack Configs налаштовуються всі драйвери. Блок конфігурації завжди

повинен бути включений у модель Simulink під час роботи з Motar. Чотири інші блоки представляють дії, які можна виконати над блоком пам'яті. Почнемо з блоку NvM Erase. Цей блок стирає вибрані блоки пам'яті після запуску. GUI блоку використовується для вибору блоку пам'яті, який слід стерти. Наступний - блок NvM Invalidate. Цей блок Simulink скасовує вибраний блок пам'яті після запуску. Блок NvM Read виводить дані, збережені у вибраному блоці пам'яті. Крім того, блок NvM Write записує введені дані до вибраного блоку пам'яті. Для всіх цих блоків виконується дія після запуску блоку Simulink, а в GUI кожного блоку слід вибирати блоки пам'яті.

Ці блоки Motar використовуються для розширення існуючої моделі керування, як показано на рисунку 1.7.

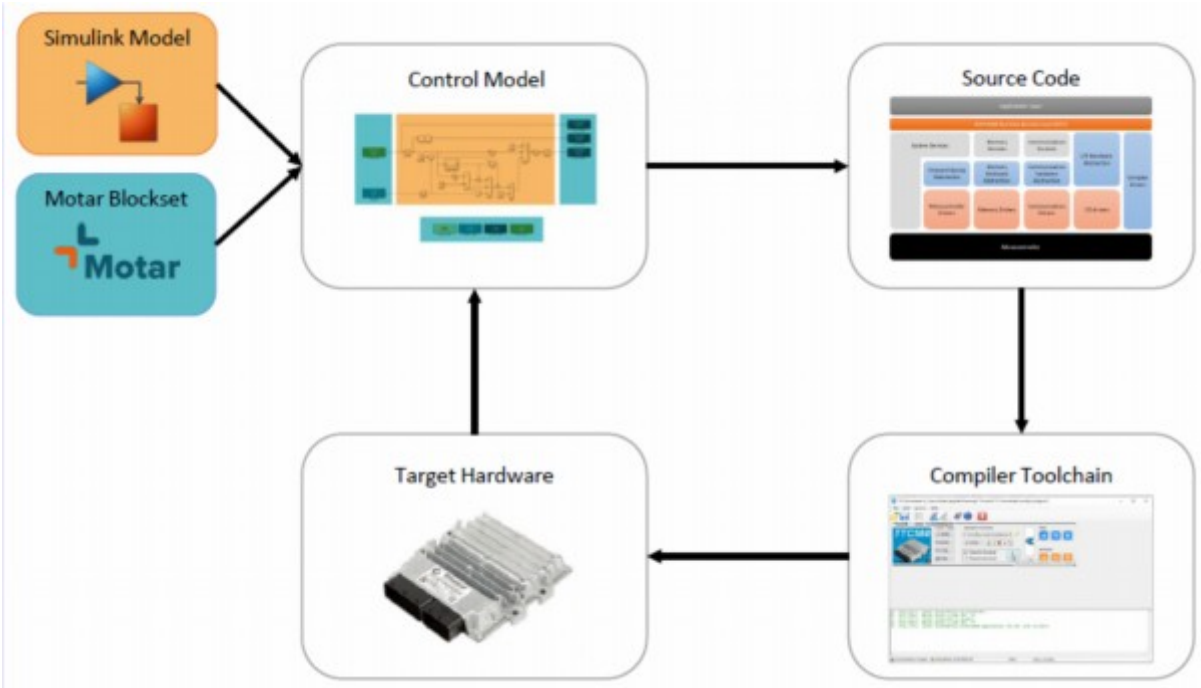

Рис. 1.7. Принцип роботи інструментарію Motar

У верхньому лівому блоці цього рисунка контрольна модель складається з помаранчевої та синьої частин. Помаранчева частина – це модель керування, створена клієнтом, а синя частина містить блоки Motar. Після розширення моделі за допомогою блоків Motar створюється вихідний

код на основі AUTOSAR. Згенерований вихідний код потім компілюється в правильну машинну мову для завантаження на цільове обладнання [3]. Коли потрібне оновлення, модель керування в Simulink можна налаштувати, щоб перезапустити цикл. Повний цикл показаний на рисунку 1.7.

1.2.2. Багатоядерні процесори

В роботі [4] обговорювалися очікувані переваги та проблеми використання багатоядерних процесорів в автомобільних додатках. Дослідження прийшло до висновку, що використання багатоядерних процесорів дає кілька переваг, таких як підвищення продуктивності, резервування безпеки та доступ до кількох архітектур. Однак він також визначив потенційні проблеми, пов’язані з впровадженням багатоядерних процесорів, особливо під час переходу існуючої системи на багатоядерний процесор.

В даний час, більше ніж через 10 років, було опубліковано безліч статей про перехід від одноядерних процесорів до багатоядерних. З цих статей стало зрозуміло, що підхід до відображення AUTOSAR Runnables може відрізнятися залежно від програми. Наприклад, у [5] представлено структуру для відображення виконуваних модулів AUTOSAR на багатоядерну платформу. Під час розробки фреймворку метою було мінімізувати загальні витрати на комунікацію. Фреймворк рішення складався з 8 кроків, результатом яких стало виконання, завдання та розподіл програм. Щоб проаналізувати результати, разом із аналізом навантаження центрального процесора (ЦП) було проведено аналіз часу. На основі аналізу навантаження ЦП було зроблено висновок, що двоядерний процесор мав збільшені накладні витрати. Однак аналіз часу показав, що двоядерний процесор міг використовувати процесор ефективніше. Таким чином, на двоядерному процесорі можна додати більше програм, щоб забезпечити краще використання процесора.

Крім зосередженості на максимально ефективному використанні процесора, ще однією метою було відображення додатків, які відповідають за безпеку. У [6] оцінено кілька різних підходів для відображення додатків AUTOSAR, які відповідають за безпеку. На додаток до дослідження щодо відображення додатків з урахуванням безпеки, було проведено дослідження можливих механізмів безпеки в багатоядерних архітектурах. У [7] запропоновано кілька механізмів безпеки для виявлення збоїв синхронізації. Ці механізми безпеки сприятимуть досягненню необхідного рівня цілісності автомобільної безпеки (ASIL), класифікації стандарту ISO 26262, який використовується для класифікації рівня небезпеки.

1.3. Особливості розробки методології для багатоядерних архітектур

У цьому розділі визначається контекст проекту. Для аналізу часу та еволюції розширення Motar Multi-Core використовується інструмент із теорії вирішення винахідницьких проблем. У [9] описується як метод вирішення проблем, заснований на винахідницькому мисленні. Інструментом, який використовується в цьому дослідженні, є діаграма дев’яти коробок [10]. Діаграма «Дев’ять коробок» — це матриця три на три, у якій стовпці представляють час, минуле, майбутнє та теперішнє. Рядки представляють різні системні рівні, підсистему, систему та суперсистему.

На рисунку 1.8 показано діаграму дев’яти коробок для проблеми, яка розглядається в цьому дослідженні. Починаючи з системного рівня, аналізується генерація коду Motar. На ранніх етапах створення коду Motar код генерувався для одноядерного процесора. Переходячи до сьогодення, Motar тепер генерує код для багатоядерного процесора. Проте лише одне ядро наразі використовується згенерованим кодом. Тому Motar потрібно буде розширити, щоб у майбутньому згенерований код використовував процесор у повній мірі.

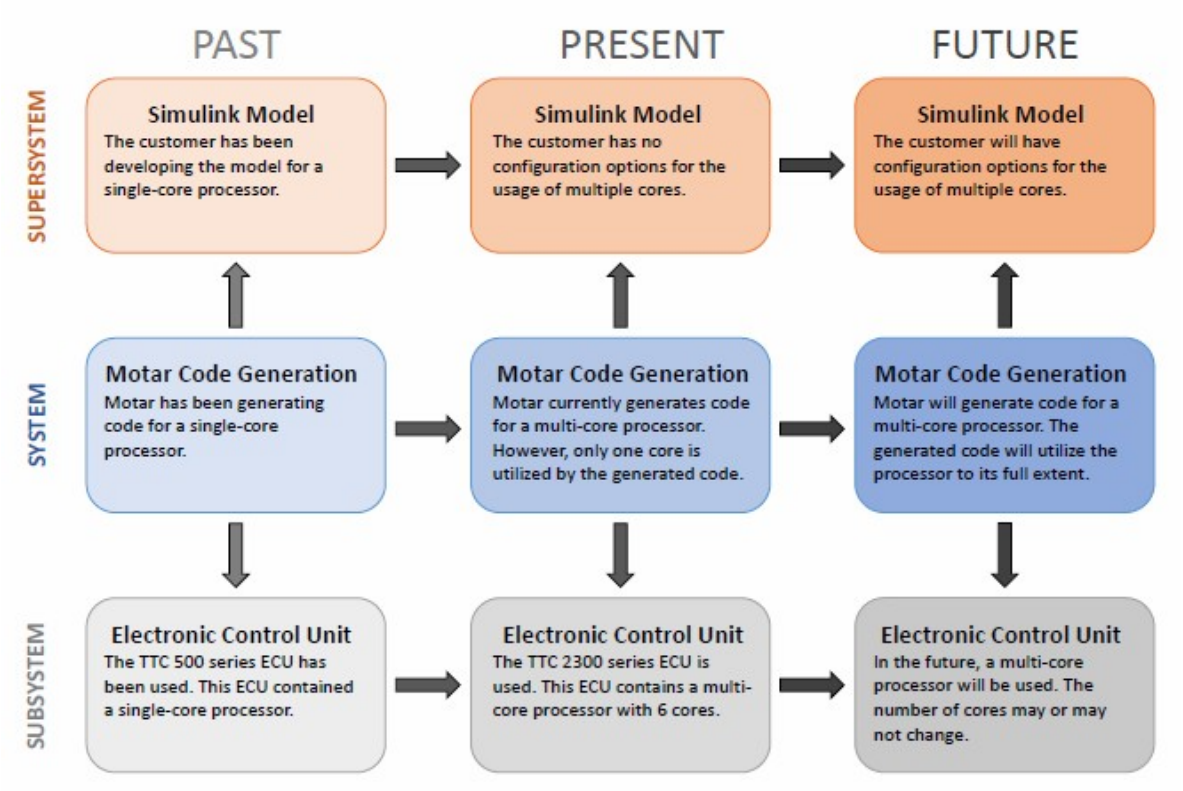

Рис. 1.9. Опис проблеми методом “дев’яти коробок”

Переходячи до підсистеми, стає зрозуміло, чому Motar повинен генерувати код для багатоядерного процесора. Як видно на рисунку 1.9, підсистема представляє цільовий ECU. У минулому цільовим ECU був TTC серії 500, який містить одноядерний процесор. Однак нещодавно з’явилася серія TTC 2300, яка містить багатоядерний процесор. У майбутньому цільовий ECU залишиться містити багатоядерний процесор. Однак кількість ядер може змінюватися, а може і не змінюватися. Тому було б корисно прагнути до рішення, яке можна адаптувати до можливої зміни ядер.

Знову дивлячись на системний рівень, також є рівень, вищий за генерацію коду Motar. У цьому проекті суперсистемою вважається розроблена замовником модель Simulink. У минулому модель Simulink була розроблена для роботи на одноядерному процесорі. Однак на сьогоднішній день це не змінилося, оскільки немає параметрів конфігурації для використання кількох ядер. Оскільки Motar краще використовуватиме процесор у майбутньому, клієнт повинен мати параметри конфігурації,

доступні в Simulink. Таким чином, клієнт матиме параметри конфігурації для використання кількох ядер, доступних у своїй моделі Simulink у майбутньому.

Щоб визначити контекст проекту, було проаналізовано обмеження цього проекту. Було визначено наступні обмеження:

- ISO 26262: Перше обмеження полягає в тому, що розширення генерації багатоядерного коду має генерувати функціонально безпечний код. Таким чином, стандарт ISO 26262 надав обмеження для розробки розширення.

- Електронний блок керування: під час цього дослідження цільовим обладнанням буде вважатися TTC 2390 ECU, що входить до сімейства TTC 2300. Тому під час розробки розширення для генерації багатоядерного коду потрібно було враховувати властивості цільового обладнання.

- Компілятор і операційна система: компілятор, який використовується для компіляції згенерованого коду, і операційна система, яка працює на цільовому обладнанні, розроблені HighTec. Таким чином, під час розробки розширення для генерації багатоядерного коду потрібно було враховувати обмеження цих сторонніх компонентів.

- Час: оскільки це дослідження є частиною магістерського проекту, існує обмеження в часі дослідження.

Для розробки розширення для створення багатоядерного коду існуючий інструментарій Motar не буде частиною системи. Існуючий набір інструментів Motar вважатиметься готовим продуктом, який впливає на розробку розширення, але не є частиною самого розширення.

1.4. Вимоги до проекту

Щоб створити чітке уявлення про мету проекту, створено вимоги до проекту високого рівня. Ці вимоги показано на рисунку 1.10. Як видно, існує одна загальна вимога до проекту, яка складається з трьох вимог. Синя вимога

охоплює аспект дослідження архітектурних шаблонів, це дослідження має вибрати найсучасніший архітектурний шаблон для включення в Motar Multi-Core Extension. Помаранчева вимога говорить про те, що під час цього дослідження має бути створено високорівневий дизайн рішення. Червона вимога говорить про те, що необхідно надати запропоноване технічне рішення.

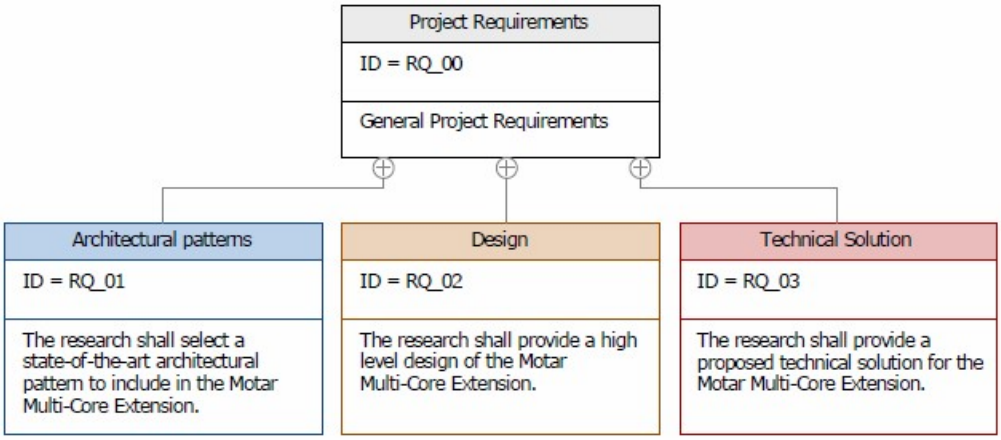

Рис. 1.10. Графічний огляд вимог проекту

Щоб визначити межі цього проекту, була створена контекстна діаграма системи. На рисунку 1.11 показано контекстну діаграму системи. У центрі діаграми контексту системи знаходиться основна система, яка є предметом цього дослідження, Motar Multi-Core Extension (MMCE). Оскільки MMCE є частиною платформи Motar, він асоціюється з Motar. На діаграмі системного контексту блок Motar представляє всі частини Motar toolbox, які вже були розроблені до цього дослідження. У групі ICT є блок під назвою developer, який представляє співробітників ICT, які розробляють Motar. Жовті блоки, які безпосередньо пов’язані з MMCE, є інструментами та компонентами, які використовує Motar, це жовті блоки. Перш за все, це блоки компілятора та операційної системи, показані внизу діаграми. Обидва ці інструменти розроблені компанією HighTec, представлені зеленим блоком. Крім того, MMCE пов’язаний з ECU, розташованим у верхній частині діаграми. ECU представляє цільове обладнання для генерації коду та розроблене TTTech

Auto. Компілятор, операційна система та ECU відіграють важливу роль у процесі генерації коду і тому пов'язані з MMCE. Іншим інструментом, пов'язаним з Motar, є Simulink. Simulink — це середовище, в якому працює Motar, і, отже, важливе для контексту системи. Останній інструмент, пов'язаний із MMCE, — це стандарт ISO 26262. Стандарт ISO 26262 описує вимоги до розробки функціонально безпечних програм. MMCE також пов'язаний із середовищем, яке представляє фактори середовища під час роботи системи.

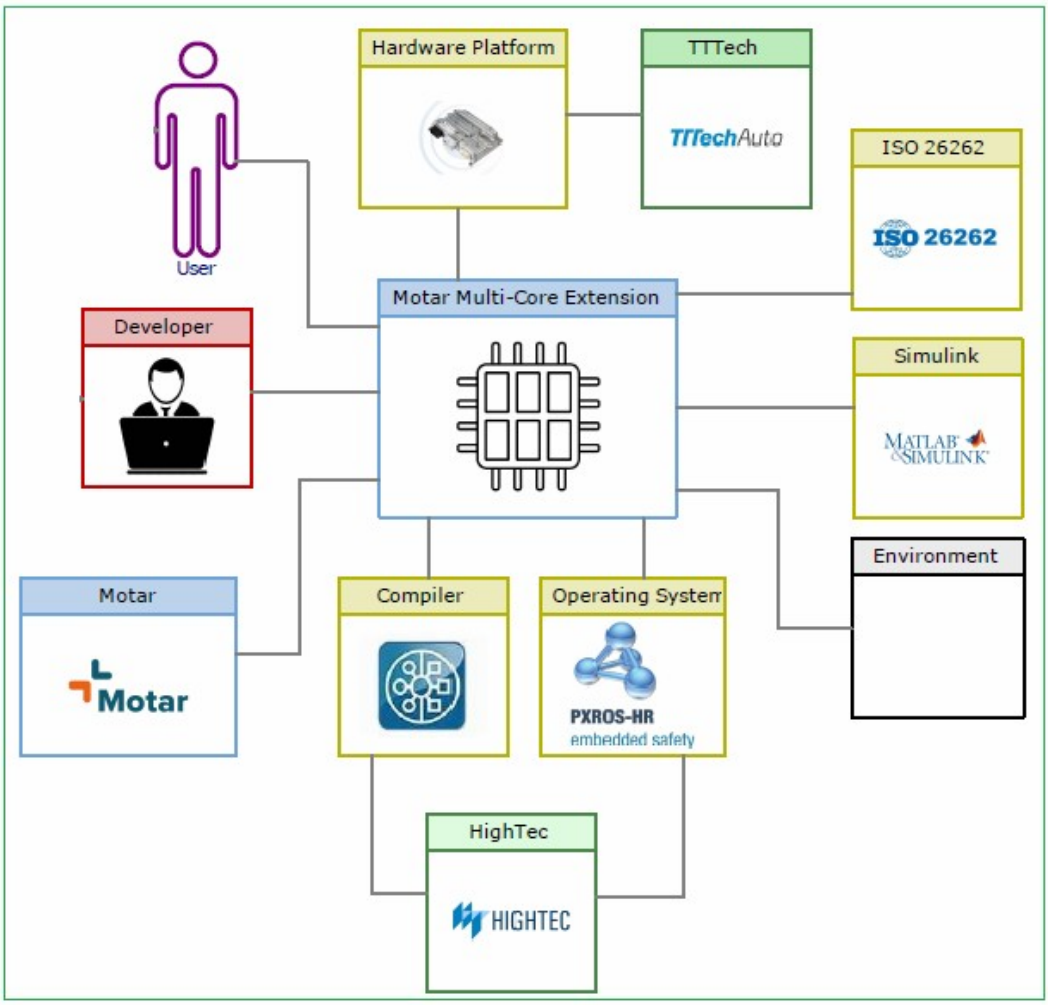

Рис. 1.11. Візуалізація контексту системи

Блоки на діаграмі системного контексту позначені кольором. Синій колір позначає основну систему, якою в цьому проекті є MMCE. Червоний колір позначає розробників MMCE, тоді як фіолетовий колір позначає

користувачів MMCE. Жовті блоки представляють інструменти та компоненти, які використовуються основною системою. Жовті блоки в деяких випадках пов’язані із зеленими блоками, які є відповідними розробниками.

Після визначення контексту системи було отримано операційне середовище системи. Операційне середовище системи забезпечує огляд суб’єктів, пов’язаних із системою під час експлуатації. Отримане операційне середовище системи показано на рисунку 1.12.

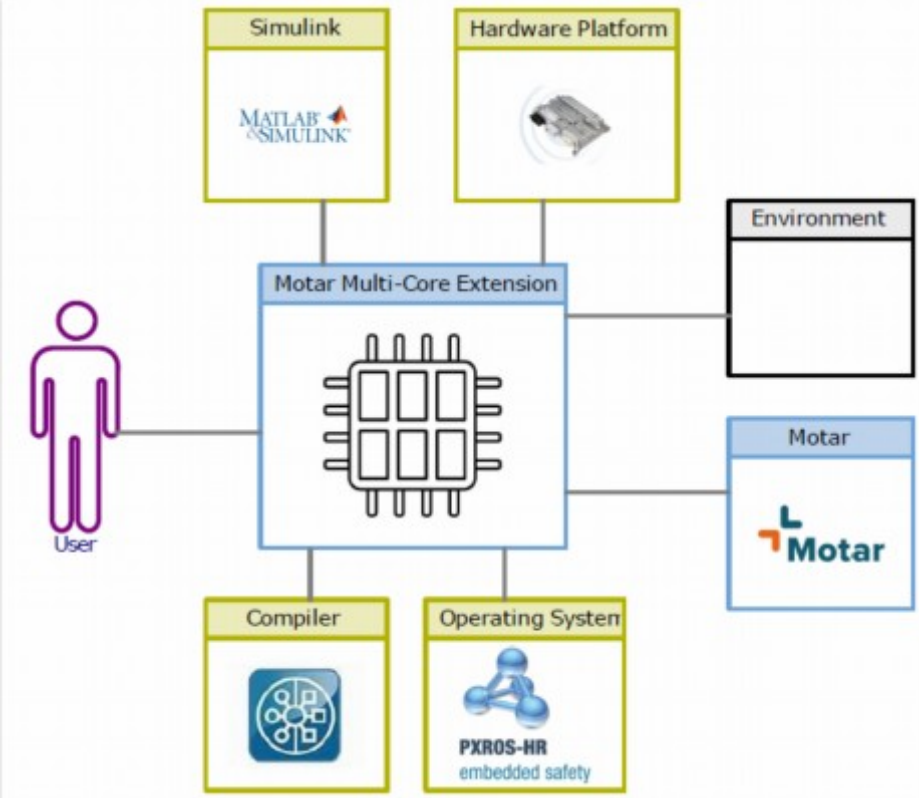

Рис. 1.12. Візуалізація операційного середовища системи

Як показано, більшість інструментів, які були визначені в контексті системи, також є частиною операційного середовища системи. Коли система використовується, Simulink є середовищем, у якому вона працює. Апаратна платформа також є частиною операційного середовища, оскільки Motar Multi-Core Extension генерує код спеціально для цільового ECU. І компілятор, і операційна система також є частиною операційного

середовища. ISO 26262 не є частиною операційного середовища, оскільки він лише надає обмеження та не впливає на систему під час роботи. Motar toolbox є частиною операційного середовища, оскільки система є розширенням Motar toolbox. Оскільки користувач керує системою, він також є частиною операційного середовища. Середовище охоплює всі фактори зовнішнього середовища і, отже, є частиною операційного середовища системи.

Висновки до розділу

У першому розділі було проведено детальний аналіз сучасного стану розробки програмного забезпечення для багатоядерних архітектур. Було визначено основні проблеми, що виникають при розробці програмного забезпечення для таких систем, а також досліджено існуючі інструментальні засоби. Особливу увагу приділено багатоядерним процесорам як основному апаратному компоненту. Крім того, було сформульовано вимоги до розробки ефективної методології для створення програмного забезпечення, що оптимально використовує можливості багатоядерних систем.

РОЗДІЛ 2. МЕТОДИ, ПРИНЦИПИ ТА ПІДХОДИ ДО ГЕНЕРАЦІЇ КОДУ ДЛЯ БАГАТОЯДЕРНИХ СИСТЕМ

2.1. Методологія проекту дослідження

Дослідження було розділено на декілька етапів, для кожного з яких методи та інструменти представлені в цьому розділі. Дослідження розпочалося з визначення проблеми, після чого було проведено попереднє дослідження. Після попереднього дослідження було створено системний аналіз Motar MultiCore Extension (MMCE). Наступні етапи технічного аналізу та технічного вирішення були зосереджені на варіанті використання розділення моделі (MPUC). Під час останнього етапу дослідження було оцінено технічний аналіз і технічне рішення MPUC. Огляд етапів дослідження показано на рисунку 2.1.

Рис. 2.1. Огляд етапів дослідження

Етап визначення проблеми складається з трьох кроків, як показано на рисунку 2.2. Першим кроком є визначення контексту проекту. На основі визначеного контексту проекту було визначено контекст системи. На останньому кроці системний контекст привів до визначення операційного середовища системи. Отримані визначення обговорювалися в розділі 1.

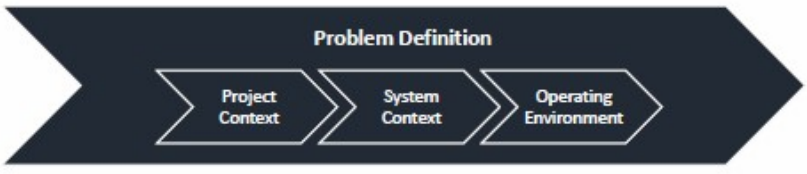

Рис. 2.2. Огляд використаної методології на етапі визначення проблеми

Щоб визначити контекст проекту, проблему було проаналізовано за допомогою діаграми дев’яти коробок. Діаграма «Дев’ять коробок» є цінним інструментом для вивчення еволюції системи з часом, а також змін у її підсистемах і суперсистемах. Щоб визначити необхідний результат цього дослідження були визначені вимоги до проекту. Визначений контекст проекту було далі визначено в контексті системи. Системний контекст – це інструмент, який є частиною методології SYSMOD [12]. Методологія SYSMOD - це набір методів і інструментів, які використовуються для розробки системи. Одним із таких інструментів є контекстна діаграма системи, в якій визначається середовище розглянутої системи. Контекстна діаграма системи дає чіткий огляд розглянутої межі системи та інтерфейсів системи. На основі створеної контекстної діаграми системи було визначено операційне середовище системи. Операційне середовище показує межі системи та систем, з якими вона взаємодіє. Визначення операційного середовища системи було останнім кроком етапу визначення проблеми. Створене визначення проблеми використовувалося на всіх інших етапах дослідження.

Наступний етап дослідження складається з двох частин, як показано на рисунку 2.3. Спочатку на етапі попереднього дослідження було проведено вивчення літератури.

Рис. 2.3. Огляд використаної методології на етапі попереднього дослідження

На основі результатів етапу попереднього дослідження було розпочато етап системного аналізу. Як показано на рисунку 2.4, систему аналізували за допомогою аналізу системних вимог, варіантів використання, архітектури

системи та поведінки системи. Ця методологія базується на SYSMOD [12]. Вимоги були визначені за результатами попередніх досліджень. Вивчення літератури дало розуміння вимог щодо функціональності системи, тоді як опитування користувачів показали вимоги щодо зручності використання системи. Результати попереднього дослідження також були використані для аналізу варіантів використання системи. На основі варіантів використання була визначена архітектура системи. Поведінка системи описує, у якому порядку виконуються визначені випадки використання.

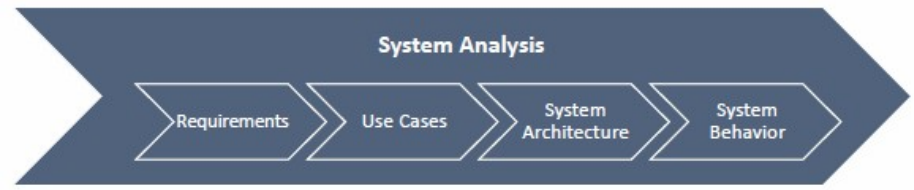

Рис. 2.4. Огляд використаної методології на етапі системного аналізу

Фаза технічного аналізу складається з трьох кроків, як показано на рисунку 2.5. Фаза розпочалася з аналізу обмежень, потім аналізу високого рівня, а потім аналізу низького рівня. Для технічного аналізу фокус більше не на MMCE, а на MPUC.

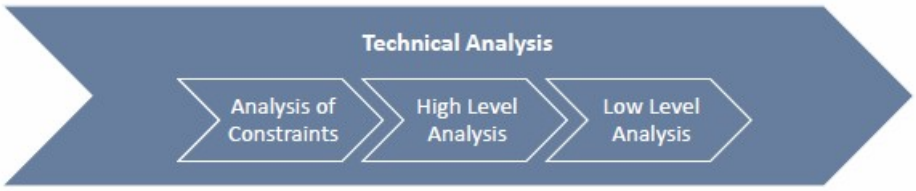

Рис. 2.5. Огляд використаної методології на етапі технічного аналізу

2.2. Дослідження стандартизованої концепції моніторингу

У 2011 році був опублікований стандарт ISO 26262 щодо функціональної безпеки дорожніх транспортних засобів. З того часу виробники автомобілів створили безліч різних архітектурних концепцій для

досягнення необхідної класифікації ASIL. Концепція E-Gas є однією з таких концепцій, яка допомагає досягти необхідної класифікації ASIL. Концепція E-Gas була вперше представлена групою з п'яти німецьких виробників автомобілів, робочою групою EGAS. Робоча група EGAS складається з BMW, Daimler, Volkswagen, Porsche та Audi. Метою цих виробників було створити стандартизовану концепцію моніторингу для систем "Drive-by-Wire" у бензинових та дизельних двигунах. Оскільки концепція E-Gas є стандартизованою концепцією моніторингу, доступна документація щодо принципів концепції.

Концепція E-Gas, як стандартизована концепція моніторингу для систем "Drive-by-Wire", передбачає використання широкого спектру методів для забезпечення безпечної та надійної роботи автомобіля. Ось деякі з найбільш поширених методів:

- Безпосередній моніторинг сигналів

- Моніторинг аналогових сигналів: Вимірювання напруги, струму, тиску та інших фізичних величин, пов'язаних з роботою системи.

- Моніторинг цифрових сигналів: Контроль стану цифрових входів та виходів, таких як сигнали датчиків, актуаторів та інших електронних компонентів.

- Аналіз частотних характеристик: Виявлення аномалій у частотних спектрах сигналів, що можуть свідчити про несправності.

- Самодіагностика

- Вбудовані тести: Проведення регулярних тестів для перевірки працездатності окремих компонентів системи.

- Моніторинг параметрів роботи: Порівняння фактичних значень параметрів з еталонними.

- Виявлення відхилень: Створення алгоритмів для виявлення відхилень від нормального режиму роботи.

- Програмне забезпечення

- Моніторинг програмного забезпечення: Контроль виконання програмного коду, виявлення помилок та збоїв.

- Перевірка цілісності даних: Захист від несанкціонованого доступу та модифікації програмного забезпечення.

- Аналіз логів: Збір та аналіз даних про роботу системи для виявлення тенденцій та прогнозування можливих проблем.

- Захист від збоїв

- Резервування: Використання дублюючих компонентів для підвищення надійності системи.

- Деградація функціональності: Збереження основних функцій системи навіть у разі відмови окремих компонентів.

- Безпечний режим: Переведення системи в безпечний режим у разі виявлення критичних помилок.

- Інші методи

- Моделювання: Створення математичних моделей системи для прогнозування її поведінки та виявлення потенційних проблем.

- Нейронні мережі: Використання нейронних мереж для виявлення аномалій, які важко виявити традиційними методами.

- Блокчейн: Забезпечення безпеки та прозорості обміну даними між різними компонентами системи.

- Комбінація методів

Концепція E-Gas передбачає використання комбінації різних методів моніторингу для забезпечення максимальної ефективності. Вибір конкретних методів залежить від типу системи "Drive-by-Wire", рівня її складності та вимог до безпеки.

Як зазначено вище, система моніторингу була розроблена для системи Driveby-Wire. Система моніторингу, яка використовувалася в концепції E-Gas, базується на 3-рівневій концепції моніторингу. На рисунку 2.6 показано графічне представлення розглянутої 3-рівневої концепції моніторингу. Рівень 1 (L1), відомий як рівень функцій, містить основні функції керування. На

додаток до основних функцій керування функціональний рівень також містить моніторинг компонентів і діагностику вхідних і вихідних змінних. Якщо на L1 виявлено несправність, система реагує відповідним чином.

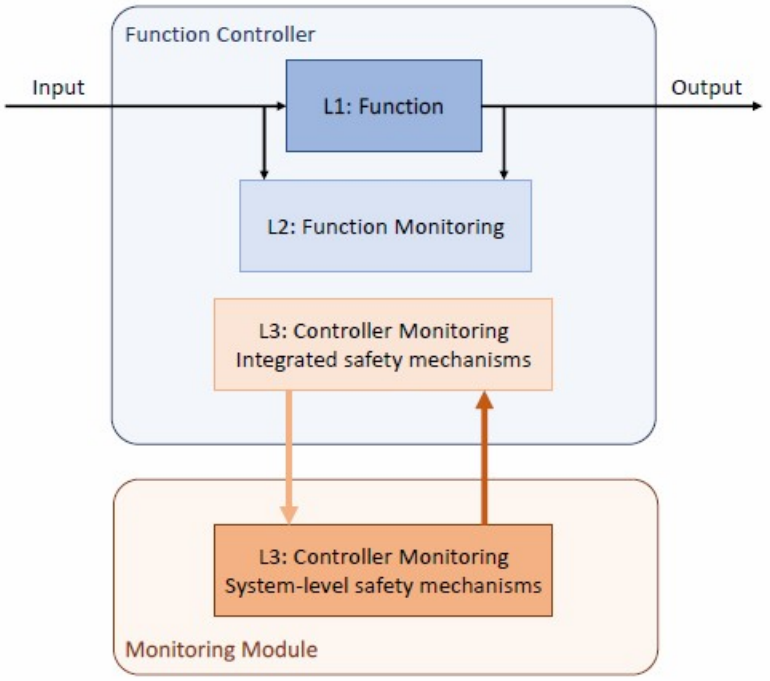

Рис. 2.6. Графічне представлення 3-рівневої концепції моніторингу

На рівні 2 (L2) контролюється функція на рівні L1. Моніторинг функціонального рівня складається з двох частин. Перш за все, рівень моніторингу функції контролює функцію, розташовану на L1. Це включає, але не обмежується моніторингом виконання елементів програмного забезпечення. По-друге, рівень моніторингу функції також порівнює вхідні та вихідні змінні з їх допустимими значеннями. Ці допустимі значення можуть бути, наприклад, діапазоном значень, який окремо обчислюється на L2. Коли виявляється несправність, L2 або діє самостійно, або ініціює реакцію на функціональному рівні.

Рівень 3 (L3) служить рівнем моніторингу контролера, враховуючи взаємодію між програмним і апаратним забезпеченням. Він дає змогу виявляти несправності функціонального контролера, наприклад несправності ядра чи пам’яті. Відповідно до [15], необхідні перевірки можуть

виконуватися як функції програмного забезпечення, апаратного забезпечення внутрішнього виявлення помилок або їх комбінація. Крім того, L3 складається як з інтегрованого механізму безпеки в контролер, так і з незалежного модуля моніторингу. Інтегрований механізм безпеки може бути програмним, апаратним або комбінованим програмно-апаратним методом виявлення. Модуль моніторингу має бути незалежним від функціонального контролера.

Окрім концепції E-gas, також деякі інші архітектури використовуються для розробки функціонально безпечних систем. У [19] порівнюється кілька архітектурних моделей, включаючи концепцію E-Gas. Частиною цього порівняння була концепція потрійного модульного резервування, яка складається з трьох ідентичних ЦП, які виконують ідентичне завдання. Виконуючи той самий код на трьох ідентичних процесорах, можна виявити можливу несправність в одному ECU [20], [21]. Однак жодного систематичного збою не виявлено, оскільки компоненти та програмне забезпечення ідентичні. Інший архітектурний шаблон, який порівнюється, – це захищений єдиний канал. У захищеній одноканальній моделі безпека покращується шляхом моніторингу вхідних даних і перевірки цілісності даних. За бажанням вихідні дані також контролюються. На підставі достовірності даних система може перейти в безпечний режим роботи, якщо це необхідно [20], [21]. Зауважте, що це не всі архітектурні шаблони, наведені в порівнянні. Однак потрійне модульне резервування, захищений єдиний канал і 3-рівневий моніторинг демонструють деякі ключові відмінності в порівнянні. Останні дві архітектури, які є частиною порівняння, — шаблон безпеки та перевірки розумності. Відповідно до [21], виконавчий орган безпеки може переключитися на вторинний канал, щоб перевести систему в безпечний стан у разі виникнення збою. Перевірка працездатності гарантує, що система працює належним чином, навіть якщо вона не зовсім коректна. Перевірка працездатності складається з компонента моніторингу, який гарантує, що активація є приблизно правильною [21].

У [22] обговорюється таке ж порівняння, як і в [19], але обговорюються ще два архітектурні шаблони. Перший з цих двох архітектурних шаблонів - це шаблон монітор-привід. Монітор Actuator — це архітектурний шаблон, на якому базується перевірка працездатності. Архітектурний шаблон складається як з каналу моніторингу, так і з каналу активації, де канал активації виконує активацію системи. Канал моніторингу перевіряє справність роботи приводу [20, 21]. Другий архітектурний шаблон порівняння, канал безпеки, був розроблений під час дослідження, яке обговорюється в [22]. Канал безпеки складається з трьох каналів: каналу приводу, каналу здоров'я та каналу кульгавого будинку. У цій архітектурі виконавчий механізм забезпечує номінальну функціональність системи. Канал працездатності контролює канал приводу та реагує в разі збою в каналі приводу.

Таблиця 2.1.

Порівняння архітектурних моделей [22]

Architectural Pattern	Reliability	Safety	Cost	Modifiability	Execution Time
Triple Modular Redundancy	R3	S3	C1	M4	T2
Monitor-Actuator	R2	S2	C3	M2	T3
Sanity Check	R2	S1	C3	M3	T3
Safety Executive	R2	S1	C2	M2	T3
Protected Single Channel	R1	S2	C3	M1	T2
3-Level Monitoring	R2	S2	C3	M2	T1
Safety Channel	R2	S2	C2	M2	T3

У порівнянні в [19] і [22] різні архітектурні шаблони порівнювалися на основі п’яти різних категорій. Перша категорія, надійність, показує відносне підвищення надійності кожного зразка. У цій категорії різні архітектурні моделі були ранжовані від R1 до R3. Якщо надійність знижується порівняно з базовою системою, архітектурний шаблон отримає квантор R1. Рівень R2

вказує на те, що надійність залишається незмінною порівняно з базовою системою, а R3 вказує на підвищення надійності. Як показано в таблиці 2.1, за винятком потрійного модульного резервування та захищеного єдиного каналу, усі архітектурні шаблони ранжуються на рівні R2. Друга категорія, безпека, вказує на покращення безпеки різних архітектурних моделей. Квантори, використані [22], вказують на низький вплив на безпеку (S1), невелике покращення безпеки (S2) або поступове покращення безпеки. Як і очікувалося, потрійне модульне резервування отримало найвищий бал у цій категорії.

Причина цих покращень безпеки така ж, як і підвищення надійності, оскільки шаблон потрійного модульного резервування продовжуватиме працювати належним чином, доки принаймні 2 канали не матимуть збою.

Третя категорія оцінювала витрати як на розробку, так і поточні витрати для кожного шаблону. Різні архітектурні моделі кількісно оцінювали як високу вартість (C1), розумну вартість (C2) або низьку вартість (C3). Через високі періодичні витрати на використання трьох різних паралельних режимів схема потрійного модульного резервування є найдорожчою. Канал безпеки та керівник системи безпеки отримали оцінку на рівні C2 через вищу вартість їх розробки, тоді як усі решта архітектурних шаблонів були оцінені як C1. У четвертій категорії оцінювалася можливість модифікації різних архітектурних моделей.

Квантор M1 використовувався, щоб вказати, що система, яка використовує відповідний архітектурний шаблон, може бути модифікована лише з додатковими витратами. Квантор M2 вказує на те, що систему можна модифікувати за допомогою простих кроків, тоді як квантор M3 вказує на те, що змінити систему дуже просто. Квантор M4 присуджується, коли архітектурний шаблон не впливає на можливість модифікації порівняно з базовою системою. Було зроблено висновок, що потрійна модульна архітектура резервування забезпечує найкращу модифікованість системи. В останній категорії, вплив на час виконання, було оцінено, наскільки

збільшився час виконання за допомогою відповідного архітектурного шаблону. Якщо спостерігалося збільшення часу виконання, архітектурний шаблон отримував квантор T1. Архітектурний шаблон, який мало впливає на час виконання, отримав квантор T2, тоді як T3 присуджувався архітектурним шаблонам, які не вплинули на час виконання. Тільки 3-рівневий шаблон моніторингу показав збільшення часу виконання. На основі порівняння в [19] і [22] можна зробити висновок, що концепція E-Gas має такі переваги:

- є стандартизованим підходом.
- не впливає на надійність.
- демонструє покращення безпеки.
- є недорогою концепцією моніторингу.
- демонструє хорошу можливість модифікації.

Однак концепція E-Gas також має недолік:

- збільшує час виконання.

2.3. Багатоядерна реалізація концепції

В [23] описані можливі багатоядерні реалізації концепції 3-рівневого моніторингу. У статті представлені дві реалізації багатоядерних контролерів і додаткова третя реалізація з використанням кількох процесорів. У цьому підрозділі підсумовано всі три запропоновані програми.

Перша програма показана на рисунку 2.7. У першому додатку рівень моніторингу функцій працює на окремому ядрі безпеки в режимі блокування. Ідея запуску двох рівнів на окремих ядрах базується на декомпозиції ASIL. Як зазначено в [24], декомпозиція ASIL – це процес поділу однієї функції на дві незалежні функції з різними ASIL. Однак рейтинги ASIL двох незалежних функцій мають дорівнювати рейтингу ASIL вихідної функції. Огляд можливих розкладів ASIL наведено в розділі 5 ISO 26262-9. Важливий Вимога до декомпозиції ASIL полягає в тому, що дві створені функції

повинні бути незалежними. Необхідну незалежність двох функцій можна підтвердити шляхом виконання аналізу залежних відмов.

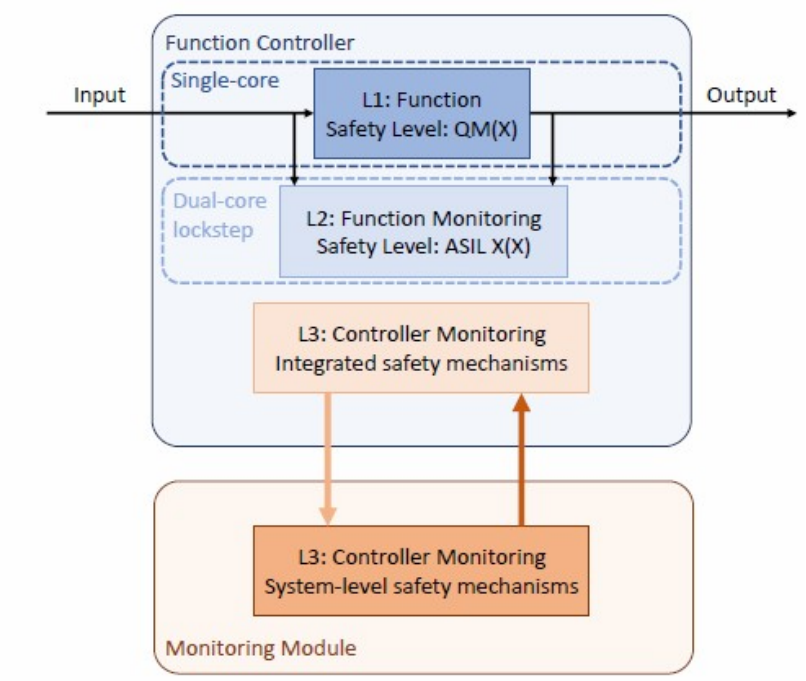

Рис. 2.7. Представлення 3-рівневого моніторингу з використанням ядра lockstep на рівні 2 (L2)

Друге застосування, яке згадується в [23], засноване на резервуванні ядер. Як показано на рисунку 2.8, обидва рівні 1 і 2 виконуються на двох різних ядрах.

Обидва ядра використовують різні периферійні пристрої, так що система може переключитися на робоче ядро, коли на одному з ядер виникає помилка. Виходи обох ядер не порівнюються один з одним, резервування використовується лише для перенесення завдань у разі збою. Таким чином, система залишатиметься працездатною для деяких помилок, які інакше призвели б до переходу системи в безпечний стан.

Третє і останнє застосування, згадане в [23], засноване на резервуванні процесорів. На рисунку 2.9 показано огляд програми, яка використовує резервування процесорів.

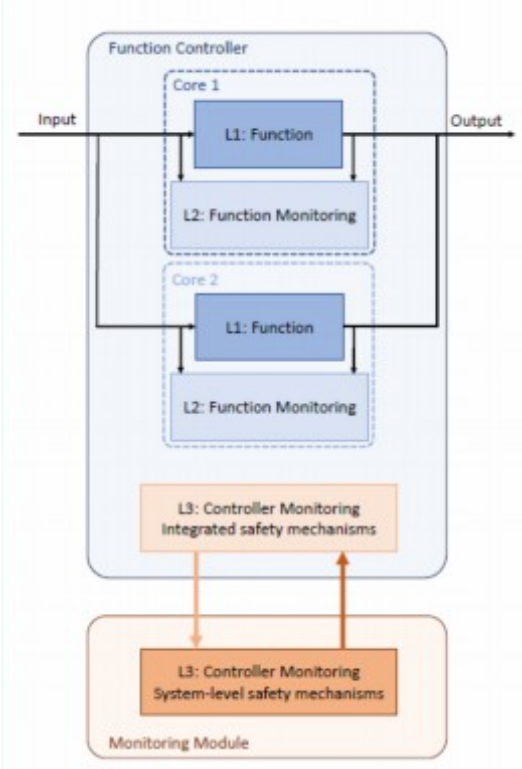

Рис. 2.8. Представлення 3-рівневого моніторингу використання резервування ядер

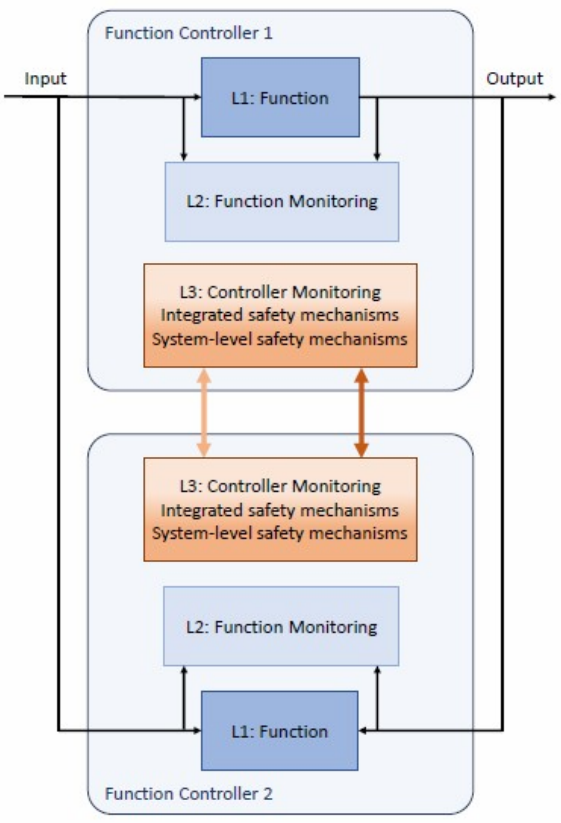

Рис. 2.9. Огляд 3-рівневого моніторингу використання резервування процесорів

Як видно, це рішення не містить зовнішнього модуля моніторингу. Замість зовнішнього модуля моніторингу системи рівня 3 обох процесорів спілкуються одна з одною для виявлення можливих несправностей. Однак, як обговорювалося в першому розділі, багатопроцесорні рішення виходять за рамки. Таким чином, ця програма більше не досліджувалась і згадується лише для надання вказівок.

2.3.1 Концепції інтерфейсу користувача

На останньому етапі дослідження представлено три різні концепції інтерфейсу користувача. Три концепції інтерфейсу користувача високого рівня показано на рисунках 2.10 – 2.12. Усі ці концепції базуються на компромісі між збереженням оригінального інтерфейсу користувача та наданням чіткого візуального огляду різних завдань.

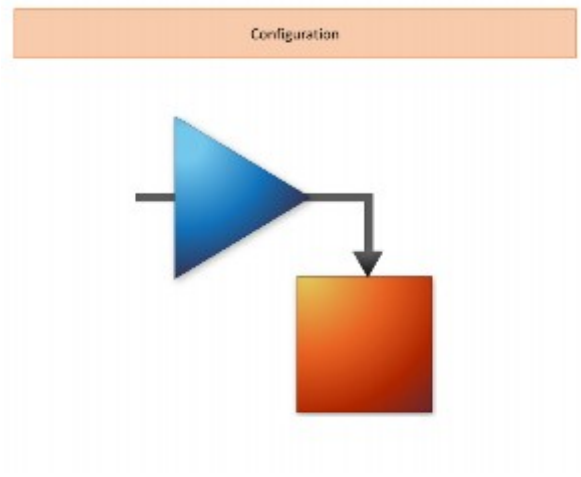

Рис. 2.10. Однорівнева концепція інтерфейсу користувача

На рисунку 2.10 показано однорівневу концепцію. Однорівнева концепція заснована на поточній версії Motar toolbox. У поточній версії набору інструментів Motar вся модель Simulink розміщена на верхньому рівні файлу Simulink. Окрім моделі, створеної користувачем, на верхньому рівні також розташована конфігурація. Однак однорівнева концепція не забезпечує чіткого огляду різних функціональних можливостей, які були змодельовані.

Концепція підсистеми завдань, показана на рисунку 2.11, використовує різні підсистеми, в які можна розмістити різні функціональні можливості.

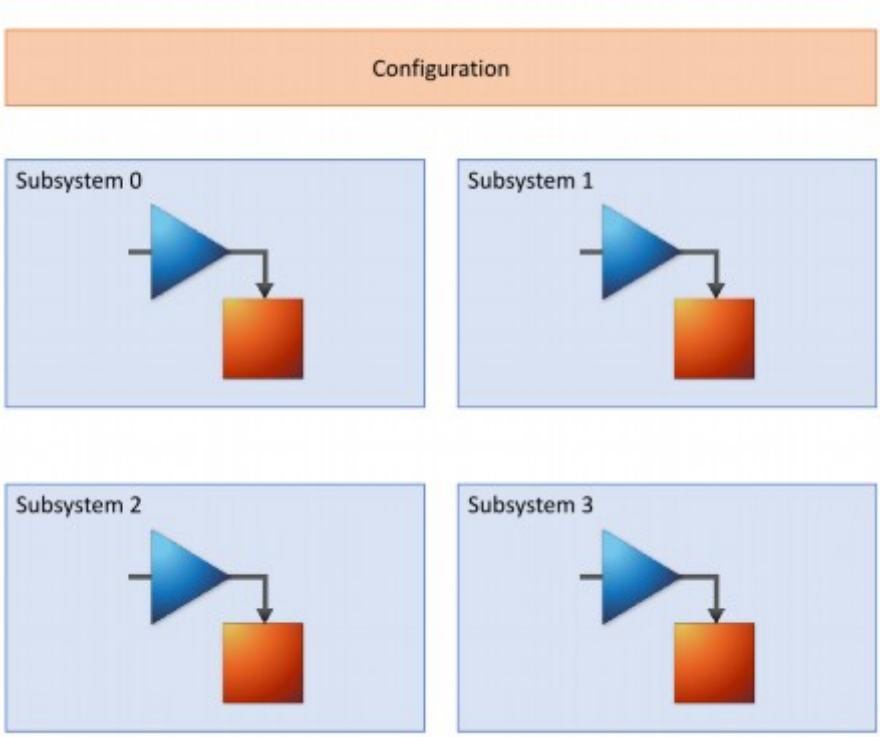

Рис. 2.11. Концепція інтерфейсу користувача високого рівня. Концепція підсистеми завдань

Це відрізняється від поточної версії Motar, де всі функції розміщено на верхньому рівні. Окрім підсистем, верхній рівень також містить конфігурацію. Концепція підсистеми завдань має на меті забезпечити чіткий огляд різних функціональних можливостей, що є одним із недоліків поточної версії Motar. Забезпечуючи чіткий огляд функціональних можливостей, концепція підсистеми завдань все ще дозволяє користувачеві налаштовувати ядра за допомогою блоків конфігурації.

На рисунку 2.12 показано концепцію багаторівневої підсистеми. Як показано, концепція багаторівневої підсистеми базується на рівні підсистем, де найвищий рівень представляє різні ядра процесора. При переході на нижчі рівні функціональні можливості можна моделювати для різних завдань, які виконуються на цьому ядрі. Концепція багаторівневої підсистеми забезпечує

більш структурований огляд, ніж концепція підсистеми завдань. Однак користувач більше не використовує блоки конфігурації для відображення завдань. Замість цього користувач повинен розмістити підсистеми в потрібному місці, щоб налаштувати відображення завдань.

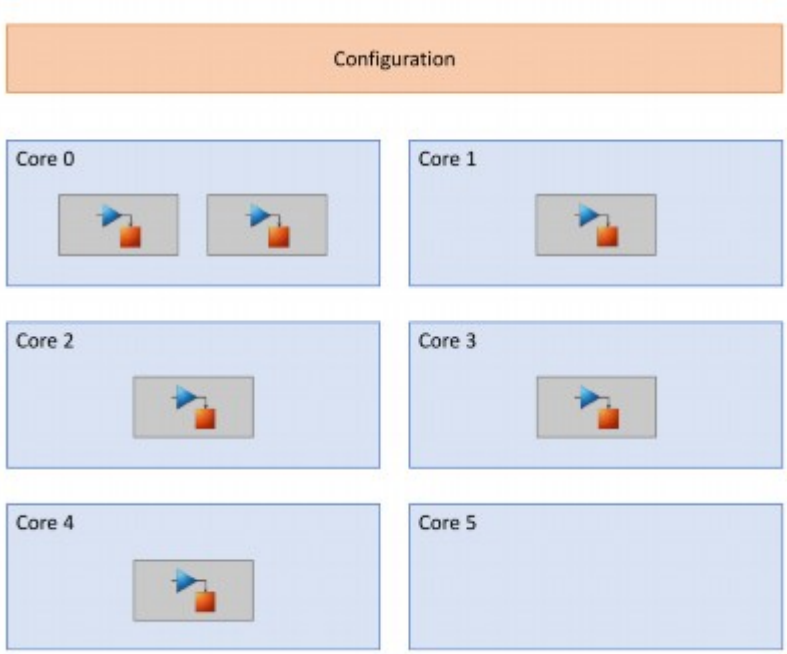

Рис. 2.12. Концепція інтерфейсу користувача високого рівня. Концепція багаторівневої підсистеми

2.4. Розробка моделі засобами Motar toolbox

Для кращого розуміння роботи Motar, було використано набір інструментів Motar toolbox для створення простої моделі. Модель представляє систему внутрішнього освітлення транспортного засобу. У цьому розділі наведено короткий опис системи та моделі Simulink.

Система внутрішнього освітлення має кілька варіантів використання, як показано на рисунку 2.13. Основний варіант використання передбачає натискання кнопки користувачем. Цей варіант розширюється двома іншими сценаріями. Після натискання кнопки внутрішнє освітлення вмикається або вимикається залежно від поточного стану системи на момент натискання.

Таким чином, система дозволяє користувачеві змінювати стан внутрішнього освітлення за допомогою кнопки. Відповідно, учасниками системи є користувач, внутрішні світильники транспортного засобу та кнопка.

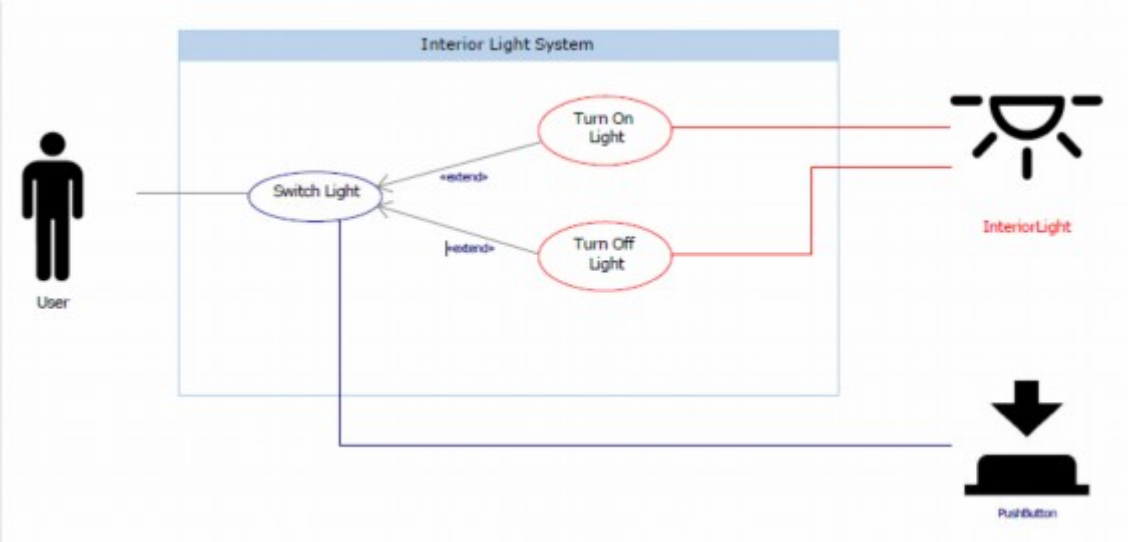

Рис. 2.13. Сценарій використання системи внутрішнього освітлення

Як показано на рисунку 2.14, модель Simulink поділена на чотири основні частини: блоки конфігурації, блоки введення, блоки виведення та модель контролера.

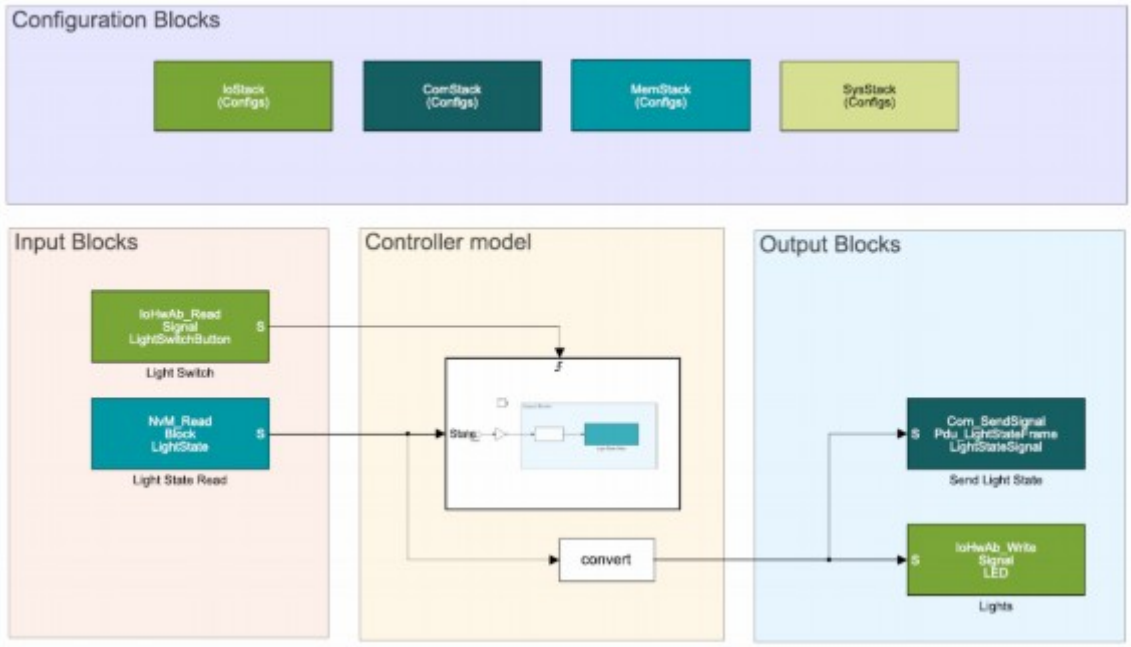

Рис. 2.14. Simulink модель системи внутрішнього освітлення

У зоні блоків конфігурації знаходяться конфігураційні блоки всіх стеків. Перший блок, IoStack Configs, визначає два сигнали. Перший сигнал, LightSwitchButton, є вхідним сигналом від кнопки. Цей сигнал налаштований з використанням підтягувального резистора (pull-down), що необхідно для того, щоб сигнал повернувся до стану "LOW" після відпускання кнопки. Другий сигнал, LED, є вихідним сигналом для керування світлодіодом (LED).

Далі, блок ComStack Configs містить налаштування шини CAN, кадру CAN та сигналу. Ці налаштування необхідні для передачі сигналу LightStateSignal через шину CAN. Третій конфігураційний блок, MemStack Configs, має налаштований блок пам'яті LightState. Цей блок використовується для зберігання стану світлодіода, що дозволяє відновити його останній стан у разі перезапуску системи. Останній конфігураційний блок, SysStack Configs, не містить конфігурацій для цієї моделі.

У зонах блоків введення та виведення ці сигнали використовуються іншими блоками Motar. У зоні блоків введення сигнал LightSwitchButton зчитується та передається до моделі контролера як тригер. Блок пам'яті LightState також зчитується, і поточний стан світильників передається як моделі контролера, так і блокам виведення. Перед тим, як використати ці дані в блоках виведення, їх тип попередньо конвертується. Вихідний блок ComStack надсилає сигнал LightStateSignal через шину CAN, який містить значення, зчитане з пам'яті. Інший вихідний блок записує значення до сигналу виходу для керування світлодіодом.

Підсистема з тригером моделі контролера представлена на рисунку 2.15. Підсистема також містить вихідний блок, який записує значення в блок пам'яті LightState. Цей блок розміщений всередині підсистеми з тригером замість зовнішнього рівня через час вибірки. Якщо розмістити цей блок на верхньому рівні, дані записуються в блок пам'яті на частоті вибірки. Розміщення його в підсистемі з тригером дозволяє записувати дані в блок пам'яті лише тоді, коли змінюється стан. Усередині підсистеми також

розміщений оператор логічного заперечення (NOT), який змінює необхідний стан.

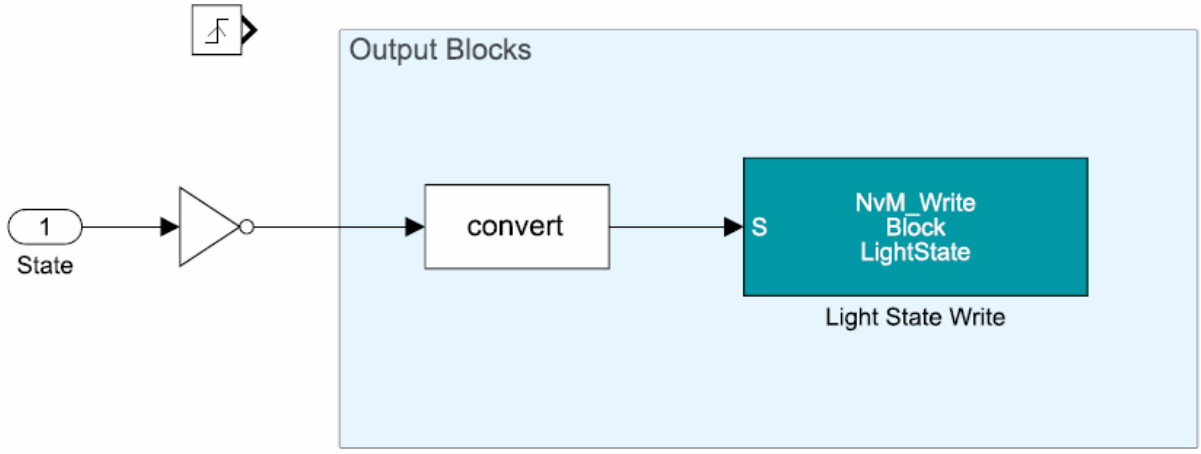

Рис. 2.15. Підсистема в моделі Simulink

Висновки до розділу

У цьому розділі розглядаються результати попереднього дослідження, яке має на меті відповісти на два підзапитання, що стосуються початкового етапу дослідження. В даному розділі висвітлюється питання використання багатоядерної архітектури для розробки програмного забезпечення, сумісного зі стандартом ISO 26262. Також розглянуто сприйняття користувачами багатоядерних архітектур у контексті використання інструментарію Motar.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ ТА МЕТОДОЛОГІЇ ГЕНЕРАЦІЇ КОДУ ДЛЯ БАГАТОЯДЕРНИХ АРХІТЕКТУР

3.1. Представлення системних вимог

Враховуючи системний контекст і результати попереднього дослідження проведеного в другому розділі, системні вимоги можуть бути визначені. Ці вимоги були розділені на чотири різні пакети вимог.

Motar Multi-Core Extension — це розширення інструментарію Motar, яке дозволяє розробляти і оптимізувати програмне забезпечення для багатоядерних архітектур в контексті автомобільних електронних блоків керування (ECU). Це розширення орієнтоване на використання можливостей багатоядерних процесорів для підвищення продуктивності, ефективності й безпеки автомобільних систем, розроблених з використанням платформи Simulink і AUTOSAR.

На рисунку 3.1 показано діаграму пакетів, яка візуалізує ці пакети та їхнє відношення до Motar Multi-Core Extension (MMCE). Кожен пакет вимог містить вимоги, пов’язані з цією категорією.

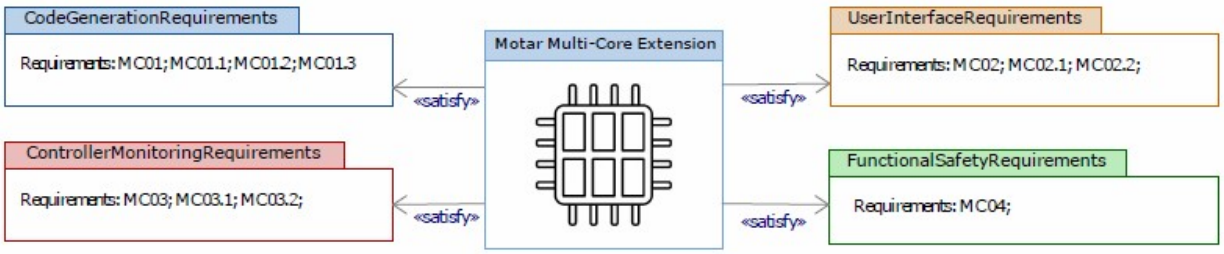

Рис. 3.1. Схема вимог Motar Multi-Core Extension

Основні можливості Motar Multi-Core Extension:

- Підтримка багатоядерних ECU. Motar Multi-Core Extension дозволяє проектувати і генерувати програмне забезпечення, яке може виконуватися на кількох ядрах процесора. Це сприяє розподілу обчислювальних навантажень між ядрами для покращення продуктивності системи.

- Розподіл задач між ядрами. Інструмент надає можливості для розподілу задач між різними ядрами. Наприклад, критичні для безпеки функції можуть бути призначені одному ядру, тоді як менш критичні задачі — іншому. Це допомагає оптимізувати використання ресурсів і знизити затримки.

- Підтримка стандартів AUTOSAR. Motar Multi-Core Extension інтегрується з AUTOSAR, підтримуючи стандартизовану архітектуру для багатоядерних систем. Це забезпечує відповідність програмного забезпечення вимогам сумісності та масштабованості.

- Оптимізація виконання. Завдяки використанню багатоядерних архітектур, програмне забезпечення може працювати паралельно, що знижує час виконання критичних операцій і покращує загальну продуктивність ECU.

- Безпека та ізоляція. Motar Multi-Core Extension дозволяє ізолювати різні функції на окремих ядрах, що покращує безпеку і надійність систем. Наприклад, системи безпеки можуть бути відокремлені від інших функцій, що зменшує ризик збоїв або перехресних помилок.

- Інтеграція з моделями Simulink. Розробники можуть використовувати Motar Multi-Core Extension для створення моделей в Simulink, що враховують специфіку багатоядерних систем. Це дозволяє проектувати й тестувати багатоядерні системи на рівні моделювання до впровадження на апаратному рівні.

Як показано на рисунку 3.1, вимоги були розділені на чотири різні категорії:

- Генерація коду (синій);
- Інтерфейс користувача (помаранчевий);
- Моніторинг контролера (червоний);

Функціональна безпека (зелений).

Ці вимоги були візуалізовані на діаграмі вимог, як показано на рисунку 3.2. Як показано на діаграмі вимог, вимоги були позначені кольором на основі відповідного пакету вимог.

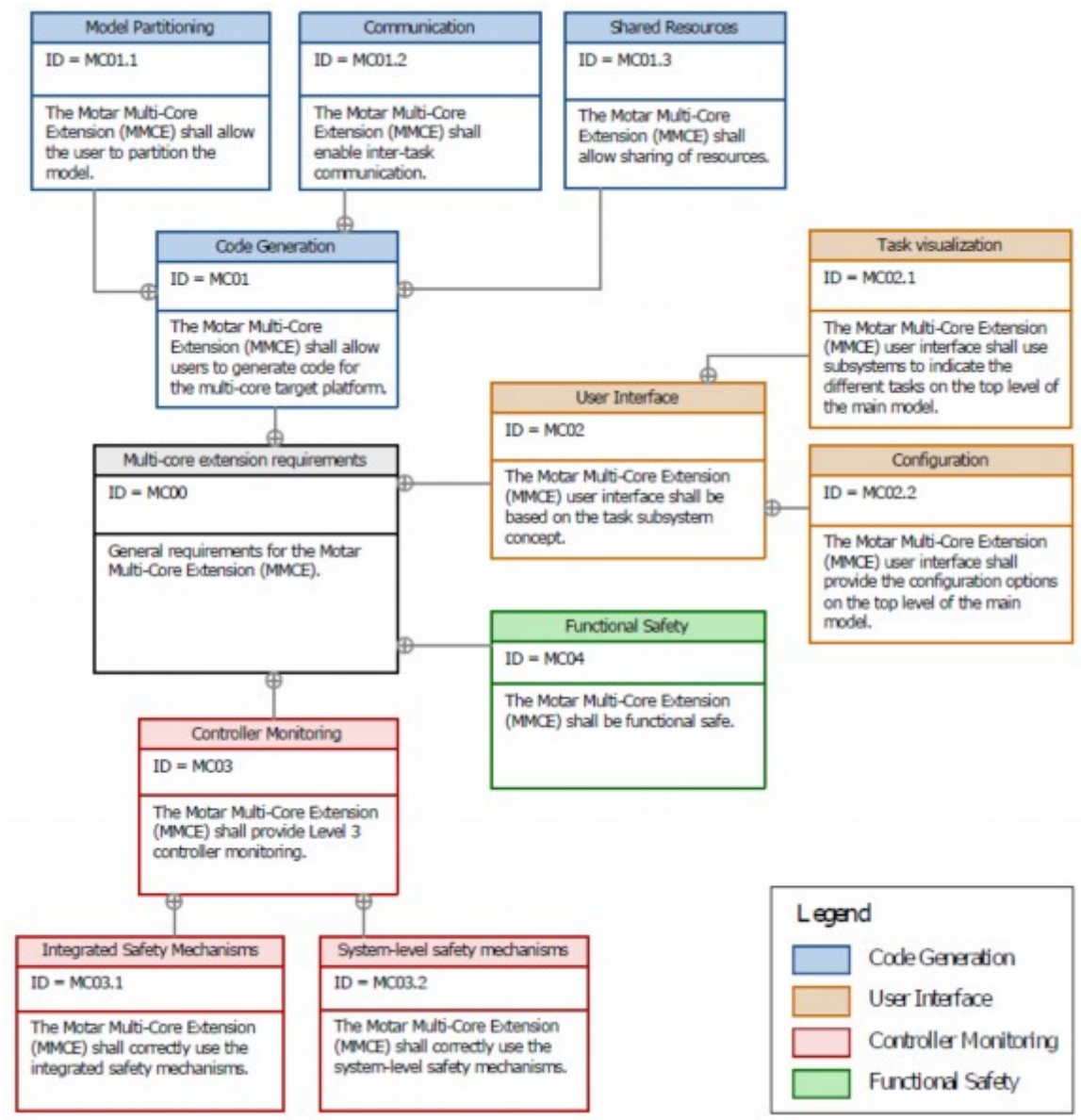

Рис. 3.2. Діаграма вимог до Motar Multi-Core Extension

Перша категорія, генерація коду, описує функціональні можливості, які MMCE має надавати для користувача. Основна вимога для цієї категорії – ID: MC01. У цій групі вимог є три додаткові вимоги, це ID: MC01.1, MC01.2 і MC01.3. Кожна з цих трьох вимог визначає більш конкретні функції, які мають бути частиною MMCE. Згідно з вимогами MC01.1, MC01.2 і MC01.3, наступні функціональні можливості повинні бути включені в MMCE:

- **Розбиття моделі Simulink:** згідно з вимогою MC01.1, розбиття моделі має бути включено в MMCE як функцію. Ця вимога випливає з багатоядерних реалізацій концепції E-Gas. У двох представлених реалізаціях

з одним контролером було вказано, на якому ядрі працюватиме кожен рівень. Однак для підтримки цього в MMCE користувач повинен мати можливість визначити, яка частина моделі відповідає якому рівню чи завданню. Таким чином, користувач повинен мати можливість розділити створену модель Simulink так, щоб можна було налаштувати різні завдання.

- **Зв'язок:** вимога MC01.2 описує, що MMCE має ввімкнути зв'язок між завданнями. Як зазначено в другому розділі, концепція E-Gas використовує концепцію 3-рівневого моніторингу. У рамках цієї концепції L2 контролює функцію, яка реалізована на L1. У деяких програмах цей моніторинг може потребувати зв'язку між завданнями для порівняння виміряних і оцінених значень. При використанні багатоядерної реалізації концепції E-Gas потрібне спілкування між різними завданнями. Таким чином, користувач повинен мати можливість використовувати зв'язок між завданнями.

- **Спільні ресурси:** вимога MC01.3 стверджує, що MMCE має дозволяти спільне використання ресурсів. У графічних представленнях концепції 3-рівневого моніторингу (розділ 2), показано, що L1 і L2 мають однакові вхідні дані. Оскільки в багатоядерній реалізації концепції E-Gas L1 і L2 є окремими завданнями, ресурс слід розподіляти між цими завданнями. Таким чином, MMCE має дозволяти розподіл ресурсів між завданнями.

Другий пакет вимог, що стосується вимог до інтерфейсу користувача. Визначено, найкращий варіант це концепція підсистеми завдань, яка показана на рисунку 2.11.. Таким чином, вимога MC02 визначає, що інтерфейс користувача MMCE має базуватися на концепції підсистеми завдань. Крім того, вимоги MC02.1 і MC02.2 містять більш детальні специфікації інтерфейсу користувача:

- **Візуалізація завдань:** як зазначено у вимозі MC02.1, інтерфейс користувача повинен використовувати підсистеми для вказівки різних завдань. Ця вимога була виведена з концепції підсистеми завдань, яка була визначена для забезпечення чіткого огляду завдань.

- **Конфігурація:** вимога MC02.2 визначає, що конфігурація для генерації коду повинна виконуватися на верхньому рівні основної моделі. Ця вимога також випливає з концепції підсистеми завдань, у якій конфігурація розташована на верхньому рівні основної моделі.

Категорія моніторингу контролера складається з трьох вимог. Основна вимога, MC03, описує, що MMCE повинен забезпечувати моніторинг контролера. Ця вимога була виведена з концепції 3-рівневого моніторингу, яка була представлена в Підрозділі 3.1.3. Для більш детальної специфікації моніторингу контролера були визначені наступні дві вимоги:

- **Інтегровані механізми безпеки:** як описано в першому розділі, моніторинг контролера L3 вимагає інтегрованих механізмів безпеки у функціональному контролері. Вимога MC03.1 визначає, що MMCE має правильно використовувати вбудовані механізми безпеки.

- **Механізми безпеки системного рівня:** окрім вбудованих механізмів безпеки, моніторинг контролера L3 також містить механізми безпеки системного рівня. Як зазначено в першому розділі, механізми безпеки системного рівня розташовані на зовнішньому модулі моніторингу. Вимога MC03.2 визначає, що MMCE має правильно використовувати механізми безпеки системного рівня.

Остання розглянута вимога, ID: MC04, визначає, що MMCE має бути функціонально безпечним. Оскільки в основному дослідницькому питанні зазначено, що згенерований код має відповідати стандарту ISO 26262, MMCE має бути функціонально безпечним.

В таблиці 3.1 містяться вимоги до MMCE.

Таблиця 3.1.

Вимоги до Motar Multi-Core Extension (MMCE)

№	ID вимоги	Специфікація
1.	MC01	Розширення Motar Multi-Core (MMCE) має дозволяти користувачам генерувати код для багатоядерної цільової платформи.

2.	MC01.1	Розширення Motar Multi-Core (MMCE) має дозволяти користувачу розділяти модель.
3.	MC01.2	Розширення Motar Multi-Core (MMCE) має забезпечувати міжзадачну комунікацію.
4.	MC01.3	Розширення Motar Multi-Core (MMCE) має дозволяти спільне використання ресурсів.
5.	MC02	Інтерфейс користувача розширення Motar Multi-Core (MMCE) має бути заснований на концепції підсистеми завдання.
6.	MC02.1	Інтерфейс користувача розширення Motar Multi-Core (MMCE) має використовувати підсистеми для позначення різних завдань на верхньому рівні основної моделі.
7.	MC02.2	Інтерфейс користувача розширення Motar Multi-Core (MMCE) має надавати опції конфігурації на верхньому рівні основної моделі.
8.	MC03	Розширення Motar Multi-Core (MMCE) має забезпечувати моніторинг контролера рівня 3.
9.	MC03.1	Розширення Motar Multi-Core (MMCE) має правильно використовувати інтегровані механізми безпеки.
10.	MC03.2	Розширення Motar Multi-Core (MMCE) має правильно використовувати механізми безпеки на рівні системи.
11.	MC04	Розширення Motar Multi-Core (MMCE) має бути функціонально безпечним.

3.2. Представлення варіантів використання

Як описано в пункті 3.1, було визначено три функціональні можливості. На основі цих вимог можна визначити варіанти використання, щоб описати, як користувач буде взаємодіяти з MMCE. Під час аналізу MMCE було визначено наступні три випадки використання:

- Модель розділення: щоб розділити модель на різні завдання, користувач повинен розділити модель. Цей варіант використання описує, як модель має бути розділена.

- Налаштування зв’язку між завданнями: для деяких програм користувачеві може знадобитися реалізувати зв’язок між різними завданнями. Комунікація між завданнями Для реалізації цього слід використовувати функцію ion. Як має бути налаштований зв’язок між завданнями, описано в цьому випадку використання.

- Налаштування спільних ресурсів: коли програма потребує спільного використання ресурсів, це можна налаштувати за допомогою функції спільних ресурсів. У цьому випадку використання описано, як мають бути налаштовані спільні ресурси.

Таблиця 3.2.

Шаблон для опису варіантів використання Motar Multi-Core Extension (MMCE)

Назва варіанту використання	
Опис	Короткий опис того, яку функціональність описує варіант використання.
Попередня умова	Стан системи перед виконанням сценарію використання.
Постумова	Стан системи після виконання сценарію використання.
Можлива помилкова ситуація	Можлива ситуація, в якій може статися помилка.
Стан системної помилки	Стан, якого досягає система у разі виникнення помилки.
Відповідні вимоги	Вимоги, пов'язані з варіантом використання.
Актори	Актори, які взаємодіють із варіантом використання.

Тригер	Подія, яка починає виконання варіанту використання.
Стандартний процес	Стандартні етапи процесу використання.
Альтернативний процес	Альтернативні етапи процесу використання.

У таблиці 3.2 показано шаблон для опису випадків використання. За допомогою цього шаблону було створено повний опис поведінки кожного випадку використання. Отримані таблиці наведено нижче. Як показано в таблиці 3.2, шаблон починається з короткого опису варіанту використання. Далі описуються передумова та післяумова. Вони містять стан системи до і після виконання варіанту використання. Крім того, у шаблоні описано можливі помилкові ситуації та відповідні їм стани системи. Для кращого відстеження відповідні вимоги також перераховані в описі варіанту використання. Огляд акторів включено, щоб описати, які актори з операційного середовища системи пов’язані з описаним варіантом використання. Тригер описує дію, яка запускає виконання описаного варіанту використання. Стандартний процес, який запускається після запуску варіанту використання, також описано в шаблоні. Якщо стандартного процесу недостатньо для правильного виконання варіанту використання, буде виконано альтернативний процес.

Таблиця 3.3.

Опис випадку використання моделі розділу

Модель розділу	
Опис	Варіант використання моделі розділу можна виконати для поділу моделі на різні завдання та налаштуйте отримані завдання
Попередня умова	Модель не розбита на кілька різних завдань.

Післяумова	Модель розділено на кілька різних завдань, і завдання налаштовано в операційній системі (ОС).
Можлива ситуація помилки	Motar Multi-Core Extension (MMCE) не може розділити модель в різні завдання.
Стан системної помилки	Motar Multi-Core Extension (MMCE) показує помилку.
Релевантні вимоги	MC01; MC01.1; MC02; MC02.1; MC02.2; MC03; MC03.1; MC03.2; MC04;
Актори	- Основні актори: Користувач; - Актори другого плану: Motar; Simulink; апаратна платформа; компілятор; операційна система;
Тригер	Користувач хоче розділити модель на різні завдання
Стандартний процес	1. Користувач розбиває модель на різні завдання. 2. Користувач налаштовує параметри кожного завдання. 3. MMCE генерує відповідний код.
Альтернативний процес	1. Користувач розбиває модель на різні завдання. 2. Користувач налаштовує параметри кожного завдання. 3. (Можливе розширення) Використання функції спілкування між завданнями виконується. 4. (Можливе розширення) Функцію використання спільних ресурсів виконано.

Таблиця 3.4.

Опис випадку використання взаємодії між завданнями

Налаштування взаємодії між завданнями	
Опис	Можна виконати сценарій використання налаштувань зв’язку між завданнями налаштувати

	зв'язок між різними завданнями.
Попередня умова	Кілька завдань налаштовано, але між ними немає зв’язку завдання.
Післяумова	Налаштовано кілька завдань і зв’язок між ними завдань налаштовано.
Можлива ситуація помилки	Motar Multi-Core Extension (MMCE) не може налаштувати спілкування.
Стан системної помилки	Motar Multi-Core Extension (MMCE) показує помилку.
Релевантні вимоги	MC01; MC01.2; MC02; MC02.1; MC02.2; MC03; MC03.1; MC03.2; MC04;
Актори	- Основні актори: Користувач; - Актори другого плану: Motar; Simulink; апаратна платформа; компілятор; операційна система;
Тригер	Зв'язок тригера потрібен принаймні між двома різними завданнями
Стандартний процес	Користувач налаштовує зв'язок між завданнями
Альтернативний процес	-

Таблиця 3.5.

Опис випадку використання спільних ресурсів конфігурації

Налаштування спільних ресурсів	
Опис	Для налаштування можна виконати сценарій використання спільних ресурсів конфігурації як розподіляти ресурси між різними завданнями
Попередня умова	Кілька завдань потребують доступу до одного ресурсу, але спільних ресурсів ще не ввімкнено
Післяумова	Завдання налаштовані на спільний доступ до ресурсу

Можлива ситуація помилки	Motar Multi-Core Extension (MMCE) не вдається налаштувати спільні ресурси
Стан системної помилки	Motar Multi-Core Extension (MMCE) показує помилку
Релевантні вимоги	MC01; MC01.3; MC02; MC02.1; MC02.2; MC03; MC03.1; MC03.2; MC04;
Актори	- Основні актори: Користувач; - Актори другого плану: Motar; Simulink; апаратна платформа; компілятор; операційна система;
Тригер	Принаймні два різні завдання повинні спільно використовувати той самий ресурс
Стандартний процес	Користувач налаштовує спільні ресурси
Альтернативний процес	-

Щоб описати зв’язки між різними варіантами використання, була створена діаграма варіантів використання. На рисунку 3.3 показано діаграму варіантів використання для MMCE. Діаграма варіантів використання показує граничну рамку для MMCE, що містить три вищезазначені варіанти використання. За межами рамки знаходяться актори та блоки, які є частиною операційного середовища системи. Як показано на рисунку 3.3, варіант використання моделі розділу розширено двома іншими варіантами використання. Оскільки виконання сценарію використання моделі розділу потрібне для створення різних завдань, користувач має виконувати лише два інші варіанти використання, щоб розширити варіант використання моделі розділу. Примітки в лівій частині діаграми вказують на умови, які описують, коли слід виконувати розширені варіанти використання.

Крім того, діаграма варіантів використання показує кілька зв’язків між обмежувальною рамкою та акторами, але жодних зв’язків між актором і варіантом використання. Це пов’язано з тим, що всі випадки використання

мають один і той же набір акторів. Оскільки всі вони мають однаковий набір акторів, асоціація з обмежувальною рамкою забезпечує кращу читабельність, ніж асоціація від кожного випадку використання до кожного актора.

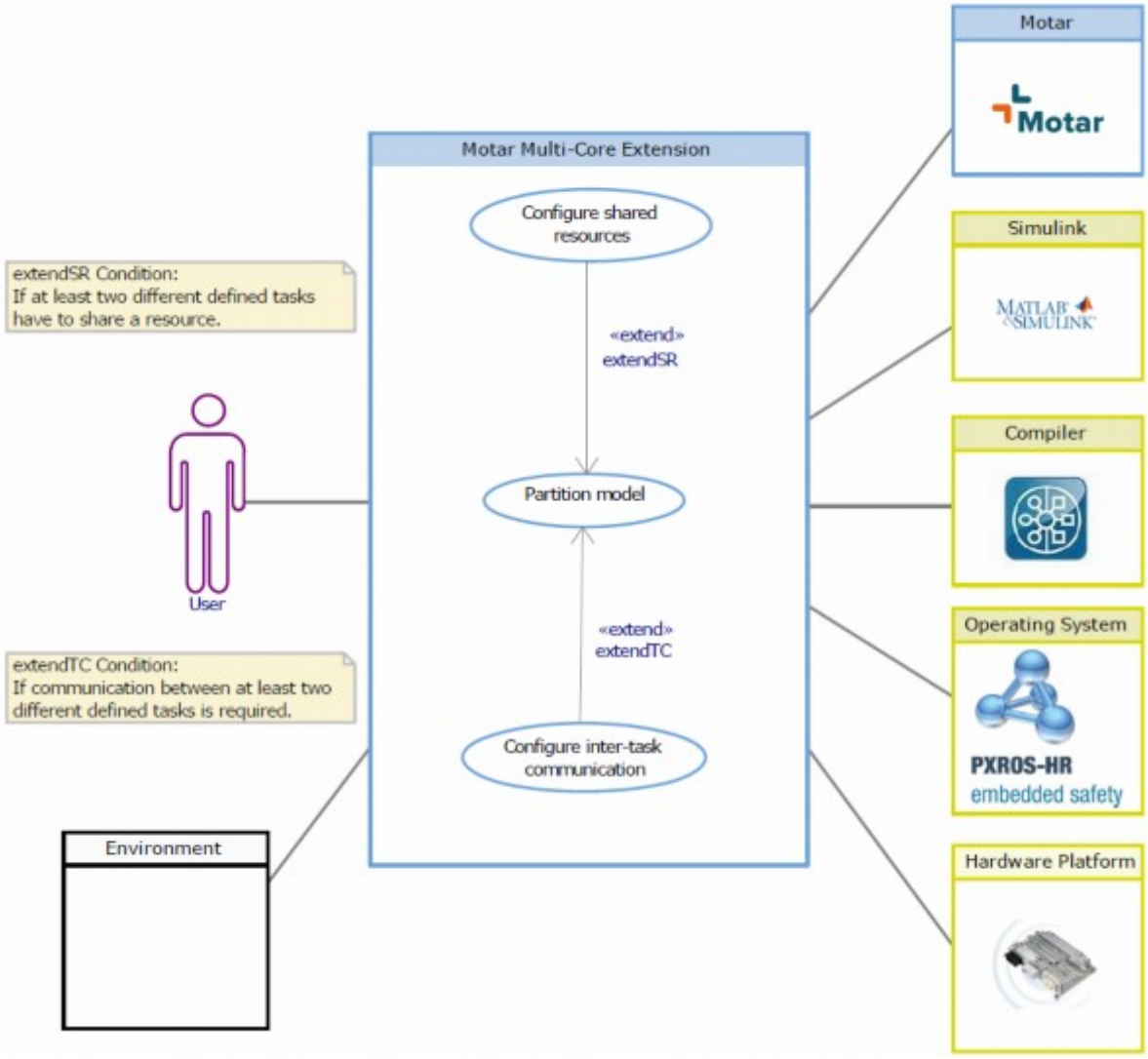

Рис. 3.3. Діаграма варіантів використання для Motar Multi-Core Extension

3.3. Реалізація архітектури системи на основі пропонованої методології

Архітектура системи – це фундамент, на якому будується вся програма. Від того, наскільки добре вона спроектована, залежить надійність, масштабованість та довговічність програмного продукту.

Основні принципи архітектури багатоядерної системи:

- Паралелізм: Завдання розбиваються на частини, і кожна частина виконується окремим ядром одночасно. Це дозволяє виконувати обчислення швидше.

- Комунікація: Ядра повинні мати можливість обмінюватися даними між собою, щоб скоординувати свою роботу.

- Синхронізація: Для того, щоб результати роботи всіх ядер були правильними, необхідно забезпечити синхронність їх дій.

- Розподіл ресурсів: Ядра спільно використовують ресурси системи, такі як пам'ять, кеш-пам'ять та периферійні пристрої.

Ефективна комунікація між ядрами є критично важливою для досягнення високої продуктивності багатоядерних систем. Вибір оптимального механізму комунікації залежить від конкретної задачі, архітектури процесора та інших факторів.

У розділі 3.2 було визначено та проаналізовано три випадки використання. Ці варіанти використання ґрунтувалися на вимогах групи генерації коду, представлених у розділі 3.1. При аналізі архітектури системи систему можна розкласти на три визначені варіанти використання, як показано на рисунку 3.4.

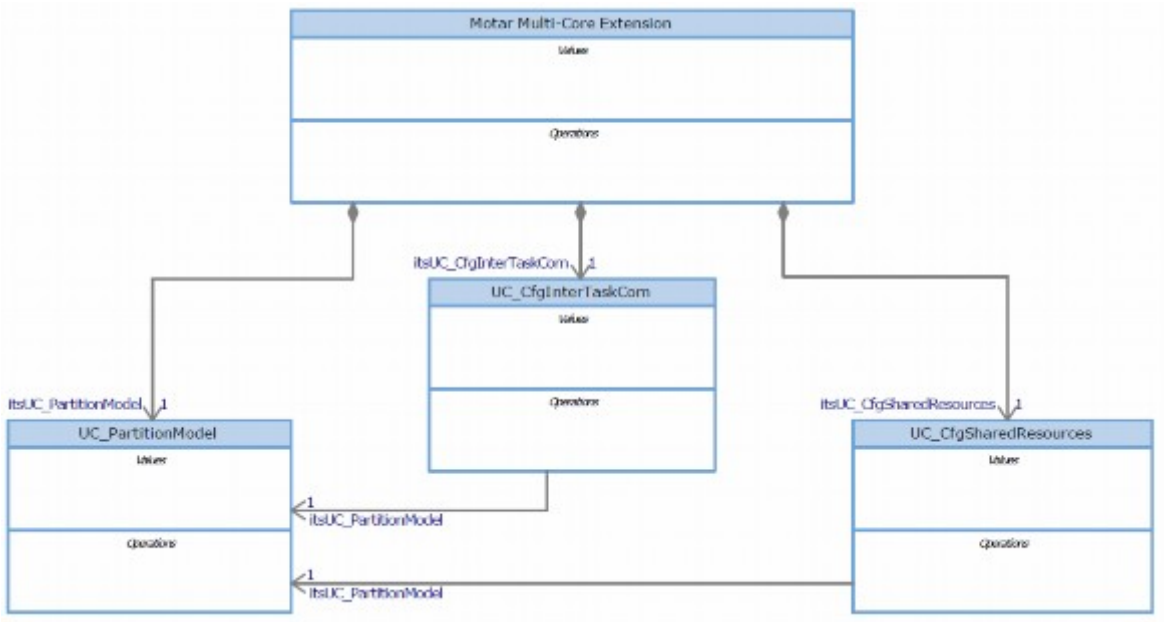

Рис. 3.4. Функції системи для Motar Multi-Core Extension

Як показано на рисунку 3.4, MMCE складається з наступних трьох програмних компонентів:

UC PartitionModel: багатоядерний процесор дозволяє обробляти кілька завдань одночасно, тоді як одноядерний процесор зможе обробляти лише одне завдання за раз. Однак для цього потрібно розділити програмне забезпечення на кілька різних завдань. Оскільки наразі неможливо налаштувати різні завдання за допомогою панелі інструментів Motar, цю функціональність потрібно включити в MMCE. Виконуючи сценарій використання моделі розділення, користувач може розділити свою модель Simulink на різні завдання.

UC CfgInterTaskCom: код, який наразі генерується панеллю інструментів Motar, складається лише з одного завдання. Тому вся комунікація наразі обробляється під одним завданням. Однак, коли модель розділена на різні завдання, модель може потребувати зв'язку між двома різними завданнями. Щоб переконатися, що завдання можуть спілкуватися одна з одною, у MMCE має бути включено варіант використання зв'язку між завданнями.

UC CfgSharedResources: оскільки поточний випуск Motar toolbox генерує код, що складається лише з одного завдання, лише одне завдання має доступ до ресурсів. Однак, коли згенерований код містить кілька завдань, деякі ресурси можуть бути спільно використані різними завданнями. Щоб забезпечити безпечний спільний доступ до ресурсів, сценарій використання налаштувань спільних ресурсів має бути включений у MMCE.

Усі три програмні компоненти, перераховані вище, є частиною MMCE. Таким чином, вони повинні будуть задовольнити вимоги інтерфейсу користувача, моніторингу контролера та пакетів вимог функціональної безпеки, які показані на рисунку 3.1.

Повна архітектурна схема системи наведена на рисунку 3.5.

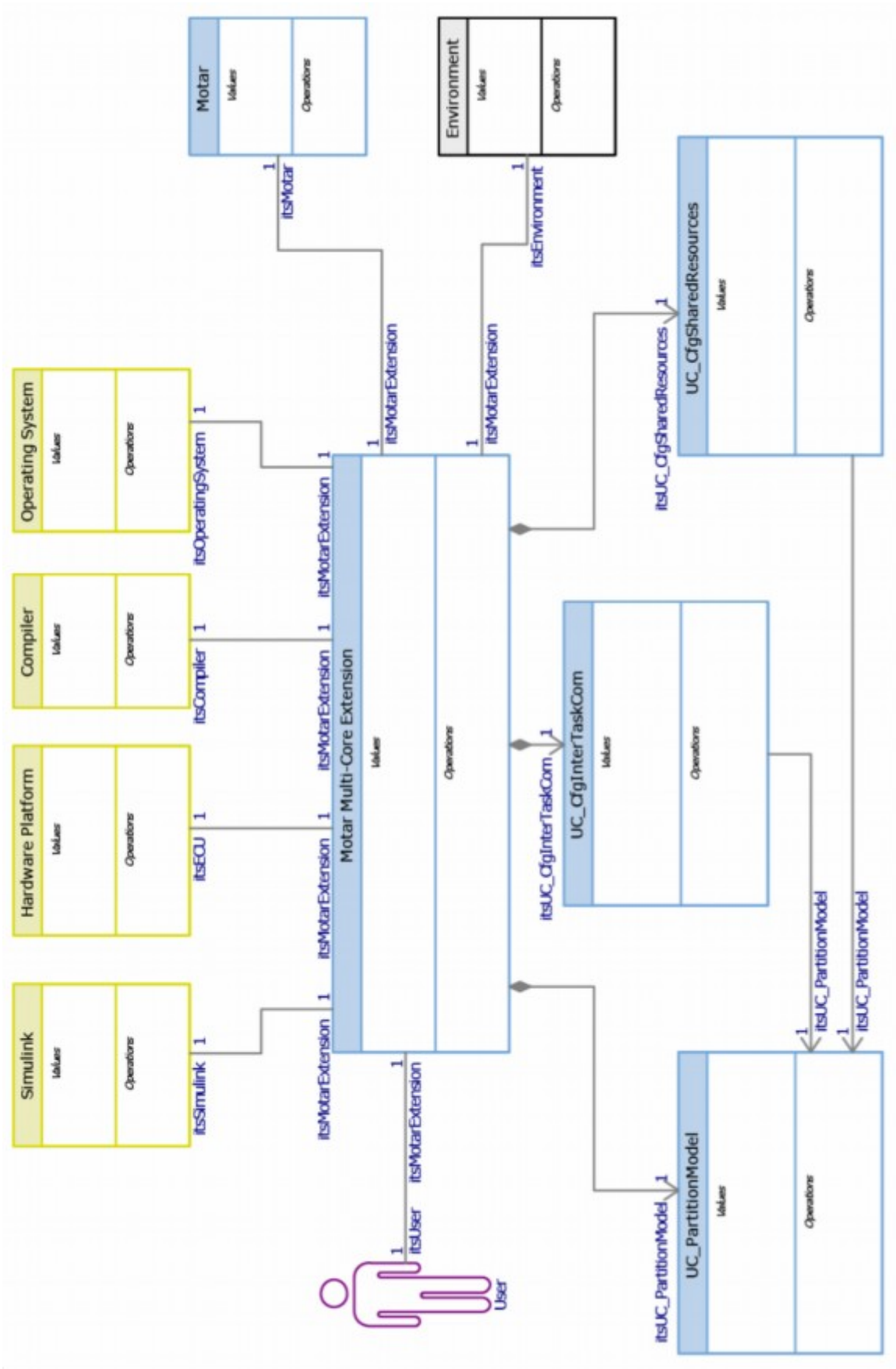

Рис. 3.5. Повна системна архітектура для Motar Multi-Core Extension (MMCE)

3.4. Розробка діаграми поведінки системи

У попередніх розділах було визначено варіанти використання та архітектуру системи. Як було показано на діаграмі варіантів використання, варіант використання моделі розділу розширено варіантами конфігурації спільних ресурсів і конфігурації міжзадачних зв'язків. Для обох цих відносин розширення було визначено умову розширення. На рисунку 3.6 нижче показано діаграму активності, яка моделює поведінку системи.

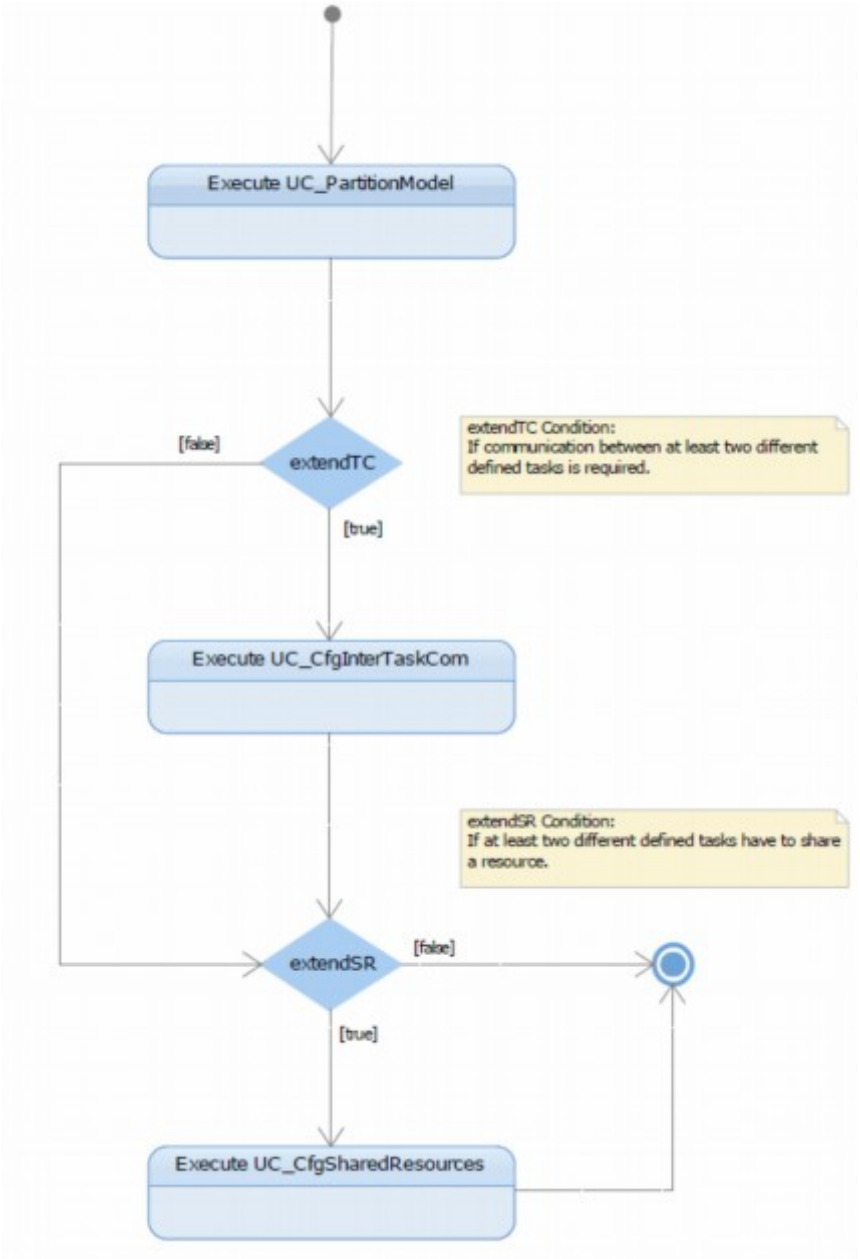

Рис. 3.6. Діаграма поведінки системи

Коли використовується MMCE, першою дією після початкової умови є виконання компонента UC PartitionModel, який відповідає варіанту використання моделі розділу. Після виконання UC PartitionModel оцінюється умова extendTC. Якщо виконується умова extendTC, виконується компонент UC CfgInterTaskCom. Якщо умова extendTC є помилковою або після виконання компонента UC CfgInterTaskCom, оцінюється умова extendSR. Якщо умова extendSR виконується, виконується компонент UC CfgSharedResources.

Висновки до розділу

У цьому розділі представлено системний аналіз побудови методології генерації коду для багатоядерних архітектур. Даний аналіз дозволяє відповісти на ключове запитання щодо розширення інструментарію Motar для створення коду, оптимізованого для багатоядерних архітектур. Системний аналіз включає чотири основні етапи. На першому етапі було визначено вимоги до системи. Після цього здійснено аналіз варіантів використання системи, що дало змогу окреслити можливі сценарії її застосування. На основі цих варіантів було побудовано архітектуру системи. Завершальним етапом стало моделювання та аналіз поведінки системи, що дозволило оцінити її ефективність і відповідність поставленим вимогам.

ВИСНОВКИ

У магістерській роботі досліджено методології генерації коду для багатоядерних архітектур. Для реалізації поставленого питання представлено вступ до тематики дослідження, викладено методологію, використану під час дослідження. Результати попереднього дослідження описуються в другому розділі, після чого представлено системний аналіз Motar Multi-Core Extension (MMCE). На основі системного аналізу, область дослідження була звужена до єдиного варіанту використання — Model Partitioning Use Case (MPUC).

Системний аналіз було проведено з метою відповісти на ключове дослідницьке запитання яке стосується розширення інструментарію Motar для генерації коду для багатоядерних архітектур.

За результатами попереднього дослідження було встановлено, що концепція E-Gas є оптимальною архітектурною схемою для розробки додатків, сумісних зі стандартом ISO 26262 з використанням Motar. Концепція E-Gas є стандартизованою в автомобільній промисловості та базується на 3-рівневій системі моніторингу. На основі висновків попереднього дослідження, під час системного аналізу були сформульовані вимоги, варіанти використання та архітектура системи. Крім вимог, для MMCE було визначено три основні варіанти використання, з яких детальнішому дослідженню було піддано Model Partitioning Use Case (MPUC).

Аналізуючи вимоги, варіанти використання та архітектуру Motar Multi-Core Extension (MMCE), було визначено три ключові функціональні можливості.

По-перше, MMCE має забезпечувати можливість розділення моделі на окремі завдання, що дозволить користувачам визначати задачі для паралельної обробки багатоядерним процесором.

По-друге, MMCE повинно підтримувати механізми спільного використання ресурсів між різними завданнями.

По-третє, MMCE має запропонувати розширення для Motar toolbox з метою забезпечення зв’язку між налаштованими завданнями. Інтерфейс користувача MMCE має бути розроблений на основі концепції підсистеми завдань для всіх визначених варіантів використання, як зазначено в третьому розділі. Додатково, MMCE має підтримувати моніторинг контролера рівня 3 та відповідати вимогам функціональної безпеки.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. J. Persoon, “ISO 26262 Certifiability of Motar: Evaluation and Future Steps.” November 2022.
2. AUTOSAR, “Layered software architecture.” https://www.autosar.org/fileadmin/user_upload/standards/classic/21-11/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf
3. ICT Group, “Motar. Reduces your time to market.” https://www.ict.eu/en/products/motar-reduces-your-time-to-market
4. P. Leteinturier, S. Brewerton, and K. Scheibert, “MultiCore benefits and challenges for automotive applications,” SAE Technical Papers, 2008.
5. P. Gupta, N. P. Singh, and G. Srinivasan, “A Framework for Real-time Automotive Applications to Multicore Platform in Perspective of AUTOSAR,” 2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings, pp. 706–709, May 2019.
6. A. Vasu and H. Ramaprasad, “Application Constraints and Safety Aware Mapping of AUTOSAR Applications on Multi-core Platforms,” 2020 IEEE International Conference on Embedded Software and Systems, ICESS 2020, December 2020.
7. A. El-Bayoumi, “ISO-26262 compliant safety-critical autonomous driving applications: Realtime interference-aware multicore architectures,” International Journal of Safety and Security Engineering, vol. 11, pp. 21–34, feb 2021.
8. Y. Dajsuren and M. Brand, Automotive Systems and Software Engineering State of the Art and Future Trends: State of the Art and Future Trends. 01 2019.
9. V. Petrov, Review of TRIZ, pp. 13–33. Springer International Publishing, 2019.

10. L. Fiorineschi, F. Frillici, and F. Rotini, “Re-design the design task through triz tools,” pp. 3–5, January 2016.
11. MindTools, “Stakeholder analysis.” https://www.mindtools.com/aol0rms/stakeholder-analysis.
12. T. Weilkiens, SYSMOD - The Systems Modeling Toolbox - Pragmatic MBSE with SysML, 2nd edition. 12 2016.
13. Statistics Solutions, “Choosing an Interview Type for Qualitative Research.” https://www.statisticssolutions.com/choosing-an-interview-type-for-qualitative-research/#:~:text=There%20are%20three%20types%20of,unstructured%2C%20semistructured%2C%20and%20structured.
14. Alyona Medelyan, “Coding Qualitative Data: How to Code Qualitative Research.” https://getthematic.com/insights/coding-qualitative-data/#:~:text=What%20is%20Deductive%20Coding%3F,also%20called%20concept%2Ddriven%20coding..
15. EGAS Workgroup, “Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units.” Version 6.0, 2013.
16. International Organization for Standardization, “ISO 26262:2018 - Road Vehicles – Functional safety - Part 3: Concept phase (ISO 26262-3:2018).”.
17. International Organization for Standardization, “ISO 26262:2018 - Road Vehicles – Functional safety - Part 5: Product development at the hardware level (ISO 26262-5:2018).”.
18. International Organization for Standardization, “ISO 26262:2018 - Road Vehicles – Functional safety - Part 6: Product development at the software level (ISO 26262-6:2018).”.
19. Y. Luo, A. K. Saberi, and M. v. den Brand, Safety-Driven Development and ISO 26262, pp. 225–254. Cham: Springer International Publishing, 2019.
20. S. P. Kumar, P. S. Ramaiah, and V. Khanaa, “Architectural patterns to design software safety based safety-critical systems,” in Proceedings of the 2011 International Conference on Communication, Computing & Security,

ICCCS ’11, (New York, NY, USA), p. 620–623, Association for Computing Machinery, 2011.

21. B. P. Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison Wesley, 2002.
22. Y. Luo, A. K. Saberi, T. Bijlsma, J. J. Lukkien, and M. Van Den Brand, “An architecture pattern for safety critical automated driving applications: Design and analysis,” 11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings, May 2017.
23. “Efficient application of multi-core processors as substitute of the E-Gas (Etc) monitoring concept,” Proceedings of 2016 SAI Computing Conference, SAI 2016, pp. 913–918, August 2016.
24. International Organization for Standardization, “ISO 26262:2018 - Road Vehicles – Functional safety - Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses (ISO 26262-9:2018).”.
25. K. van Bergeijk, “Motar platform HY-TTC 500 family v1.5.” August 2022.
26. Asanovic, K., et al. "The Landscape of Parallel Computing Research: A View from Berkeley." Technical Report, University of California, Berkeley, 2006.
27. Lee, E. A., and Sangiovanni-Vincentelli, A. "A Unified Framework for Comparing Models of Computation." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1998.
28. Kaxiras, S., and Martonosi, M. "Computer Architecture Techniques for Power-Efficiency." Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2008.
29. Chandra, R., et al. "Parallel Programming in OpenMP." Morgan Kaufmann Publishers, 2001.
30. Olukotun, K., and Hammond, L. "The Future of Microprocessors." ACM Queue, 2005.
31. Ghosh, S., et al. "Efficient Cache Coherence in Multicore Architectures." Journal of Parallel and Distributed Computing, 2010.

32. Jouppi, N. P., et al. "High Performance Data Cache Architecture." IEEE Transactions on Computers, 2001.
33. Xue, J., et al. "Automatic Parallelization for Multicore Processors." ACM Transactions on Architecture and Code Optimization, 2011.
34. Puschner, P., and Burns, A. "A Review of Worst-Case Execution Time Analysis." Real-Time Systems, 2000.
35. Sarkar, V. "Optimized Code Generation for Multicore Architectures." ACM Transactions on Programming Languages and Systems, 2005.
36. Sutter, H. "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software." Dr. Dobb's Journal, 2005.
37. Mattson, T. G., et al. "Patterns for Parallel Programming." Addison-Wesley Professional, 2004.
38. Drepper, U. "What Every Programmer Should Know About Memory." Red Hat, Inc., 2007.
39. Aigner, W., and Miksch, S. "Visualization of Time-Oriented Data in Multicore Systems." IEEE Transactions on Visualization and Computer Graphics, 2004.
40. Kessler, C. W. "Code Generation for Embedded Multicore Architectures." Journal of Systems Architecture, 2009.
41. Beyls, K., and D'Hollander, E. H. "Reuse Distance as a Metric for Cache Behavior." Journal of Systems Architecture, 2001.
42. Grun, P., et al. "A Framework for Compiler-Directed Exploration of Architectural Features for Coarse-Grain Parallel Processors." ACM Transactions on Design Automation of Electronic Systems, 2001.
43. Dubey, P. "Recognition, Mining, and Synthesis Moves Computers to the Era of Tera." Technology@Intel Magazine, 2005.
44. Peleg, A., and Weiser, U. "MMX Technology Extension to the Intel Architecture." IEEE Micro, 1996.

45. Muller, H., et al. "Automatic Parallelization of Nested Loops for Multicore Architectures." ACM Transactions on Architecture and Code Optimization, 2012.
46. McCool, M., et al. "Structured Parallel Programming: Patterns for Efficient Computation." Morgan Kaufmann Publishers, 2012.
47. Pop, P., et al. "Analysis and Optimization of Real-Time Systems with Multicore Processors." ACM Transactions on Embedded Computing Systems, 2004.
48. Carr, S., and Kennedy, K. "Improving the Ratio of Communication to Computation in Coarse-Grain Parallel Programs." ACM Transactions on Programming Languages and Systems, 1995.
49. Allen, R., and Kennedy, K. "Optimizing Compilers for Modern Architectures: A Dependence-Based Approach." Morgan Kaufmann Publishers, 2002.
50. Chandy, K. M., and Taylor, S. "An Introduction to Parallel Programming." Jones and Bartlett Publishers, 1992.