Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделювання апаратного забезпечення з використанням автоматичної генерації коду за допомогою MATLAB/Simulink
(тема)

Виконав:

студент II курсу, групи СПзм-22-1

Капахі Анжеліка
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: ст.викладач Знайдюк В.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Капахі Анжелікі
(прізвище, ім'я, по батькові)

1. Тема роботи Моделювання апаратного забезпечення з використанням автоматичної генераціїкоду за допомогою MATLAB/Simulink

затверджена наказом по університету від " 01 " квітня 2024 р. № 45 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи Моделювання апаратного забезпечення
Автоматична генерація
MATLAB/Simulink

4. Перелік питань, що потрібно опрацювати у роботі
Загальні положення
Пропонований робочий процес проектування
Характеристика в BUCK-перетворювачі
Валідація в більш складній схемі
Результати експерименту
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 12

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд літературних джерел	01.04.24-08.04.24	
2	Вибір та обґрунтування методики та засобів	09.04.24-12.04.24	
	дослідження		
3	Вибір та обґрунтування методів	13.04.24-19.04.24	
4	Програмна реалізація проєкта	20.04.24-09.05.24	
5	Проведення експериментальних досліджень	10.05.24-23.05.24	
6	Оформлення матеріалів кваліфікаційної роботи	24.05.24-03.06.24	
7	Подання кваліфікаційної роботи керівникові та	04.06.24-07.06.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на	08.06.24-12.06.24	
	рецензування		
9	Захист		

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст.викладач Знайлюк В.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 50 с., 11 рис., 13 табл., 1 дод., 23 джерел.

HARDWARE-IN-THE-LOOP, MATLAB, SIMULINK, FPGA; ПЕРЕТВОРЮВАЧІ ЕНЕРГІЇ

Метою кваліфікаційної роботи є запропонувати робочий процес для моделювання апаратного забезпечення з використанням автоматичної генерації коду за допомогою MATLAB/Simulink

У ході виконання кваліфікаційної роботи проведено ґрунтовний аналіз сучасних методів моделювання і генерації коду, розроблено ефективні моделі апаратного забезпечення, проведено симуляції і експерименти, які підтверджують ефективність запропонованих методів. Запропоновано модифікацію робочого процесу проектування HIL-моделей, яка призводить до автоматичної генерації дуже ефективних моделей. Це усуває необхідність проектування на рівні HDL і забезпечує результати, дуже близькі до результатів ручного проектування.

ABSTRACT

Master’s thesis: 50 pages, 11 figures, 13 tables, 1 appendices, 23 sources.

HARDWARE-IN-THE-LOOP, MATLAB, SIMULINK, FPGA; POWER CONVERTERS

The major goal of this thesis is to propose a workflow for hardware simulation using automatic code generation using MATLAB/Simulink

In the course of the qualification work, a thorough analysis of modern modeling and code generation methods was carried out, effective hardware models were developed, simulations and experiments were carried out, which confirm the effectiveness of the proposed methods. A modification of the HIL model design workflow is proposed, which leads to the automatic generation of highly efficient models. This eliminates the need for design at the HDL level and provides results very close to those of manual design.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MATLAB – Matrix Laboratory, програмне забезпечення для математичних розрахунків і моделювання.

Simulink – Інструмент моделювання і симуляції, інтегрований з MATLAB.

Автоматична генерація коду (AGC) – Процес автоматичного створення програмного коду на основі моделі, створеної в MATLAB/Simulink.

Апаратне забезпечення (АЗ) – Фізичні компоненти системи, які взаємодіють з програмним забезпеченням для виконання певних завдань.

Моделювання апаратного забезпечення (МАЗ) – Процес створення математичної моделі апаратних компонентів для проведення їхньої симуляції і тестування.

ВСТУП

Тестування апаратного забезпечення в циклі зазвичай є частиною циклу проектування систем керування. Ефективні та швидкі моделі можуть бути створені на мові опису апаратного забезпечення (HDL – Hardware Description Language), яка реалізована в програмованій матриці вентилів (FPGA – Field-Programmable Gate Array).

Інженери з управління більш досвідчені в підходах більш високого рівня. HDL-моделі, отримані автоматично зі схем, мають помітно нижчу продуктивність, тоді як HDL-моделі, отримані на основі рівнянь, швидші та менші за розміром. Однак навіть моделі, автоматично перекладені на HDL за допомогою рівнянь, можуть бути гіршими, ніж моделі, створені вручну.

Пропонується робочий процес проектування, який дозволяє досягти продуктивності, подібної до ручної роботи, за допомогою автоматичних інструментів. Він складається з ідентифікації подібних операцій, примусової перевірки знаковості сигналів і пристосування до розмірів вхідних даних множників. Проведено детальне порівняння між трьома робочими процесами: переклад високорівневого коду MATLAB, переклад моделі Simulink і робота безпосередньо в HDL.

Джерела неефективності були показані в buck-перетворювачі, а процес був перевірений в повному мості з електричними втратами за допомогою методу Рунге-Кутта. Результати показали, що запропонований підхід забезпечує код, який працює дуже близько до еталонної реалізації на VHDL, навіть для складних конструкцій. Нарешті, модель було реалізовано на готовій платі ПЛІС, придатній для апаратного тестування в циклі.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

Методи Hardware-In-the-Loop (HIL) значно полегшують етапи тестування складних систем, оскільки вони мінімізують ризик травм або поломки обладнання [1,2]. У цих методах один з елементів реальної системи замінюється моделлю, яка працює в реальному часі. Модель установки забезпечує такий самий зворотній зв'язок з контролером, як і реальний елемент. У силовій електроніці HIL-модель забезпечує безпечне тестування без використання частин, що знаходяться під напругою, доки конструкція не буде достатньо протестована [3]. Вона також дозволяє тестувати контролери на віртуальній установці перед створенням перетворювача.

Моделі працюють в режимі реального часу і повинні розраховувати свої стани з високою точністю. Можна використовувати або комерційні системи (такі як Opal-RT, dSPACE і Typhoon HIL), або пристрої на основі програмованих користувачем вентильних матриць (FPGA – Field Programmable Gate Array) [4,5,6], запрограмовані на мові опису апаратного забезпечення (HDL), наприклад, Verilog або VHDL.

Використання елементів HIL є одним з етапів V-подібного циклу розробки складних систем [1]. Інженер-енергетик зазвичай будує модель мовою високого рівня. Ручний переклад у HIL-реалізацію вимагає навичок HDL, що означає додаткові зусилля або використання автоматичних інструментів.

Кілька робіт показали використання комерційних інструментів для генерації високопродуктивних HIL-моделей на ПЛІС: електричних і механічних компонентів системи перетворення енергії вітру в [7], фотоелектричної системи в [8] або електричних мереж в [9].

У цих випадках елементи обробки генеруються за допомогою автоматизованих робочих процесів, але автори не аналізують, наскільки вони далекі від найкращої продуктивності, яку зазвичай отримують при ручному

проектуванні на HDL. У [10,11] автори стверджують, що автоматична генерація є швидшою і менш схильною до помилок для розробників силової електроніки, хоча порівняння з моделлю, згенерованою вручну, не проводилося. У роботах [12,13,14] таке порівняння було зроблено, і автори показали, що автоматизована модель використовує більше ресурсів, ніж ручна модель; однак вони не проаналізували джерело цієї невідповідності. У роботі [15] автори показали, що модель, згенерована автоматично, потребує більш ніж удвічі більшої кількості елементів-співмножників, не надаючи жодних пояснень. Були запропоновані деякі робочі процеси, такі як [16], але вони орієнтовані на фахівців з ПЛІС, і в них не перевіряється, чи згенерований VHDL-код є оптимальним. На етапі генерації VHDL-коду [16] після автоматичного перекладу не було виконано жодної перевірки якості перекладу.

Досі вибір для дизайнера полягав у тому, щоб отримати або дуже ефективну модель за допомогою ручного проектування, або гіршу – за допомогою автоматичного перекладу. Перший варіант вимагає великих зусиль і навичок роботи з HDL. Другий варіант вимагає менше зусиль і часу, але може призвести до неоптимальних моделей. Це може бути неочевидним, оскільки додаткове використання ресурсів може бути компенсоване використанням потужнішого обладнання. Однак, зрештою, це означає, що або HIL-моделі завищують вимоги до апаратного забезпечення, або використовується більша, ніж потрібно, частина наявного апаратного забезпечення, забираючи ресурси, які можна було б присвятити, наприклад, підвищенню точності симуляції. Кінцевим наслідком є вища вартість моделювання, втрата точності або навіть і те, і інше.

У цій роботі пропонується робочий процес для автоматичної генерації моделей, який має на меті досягти найкращих результатів автоматичного та ручного підходів: висока ефективність, низькі зусилля, легке додавання інтерфейсів, відсутність необхідності програмування на низькорівневій мові HDL та сумісність зі складними моделями та різними чисельними методами.

Як показано на рисунку 1.1, для побудови ефективної моделі HIL перший вибір – між комерційною та спеціальною системами. Спеціальні системи пропонують найкращу продуктивність та гнучкість. У них модель описується або на рівні схеми, або на рівні рівняння стану. Бібліотека MATLAB/Simulink Simscape дозволяє моделювати на рівні схеми. Це менш вимогливо, але пропонує менше контролю для проектувальника.

Рисунок 1.1 – Дерево рішень для впровадження режиму HIL

Високорівневий синтез і G/LabView вимагають менше зусиль, але, як показано в [17], їхні результати не були задовільними. Високорівневий синтез призводить до високого навантаження ресурсів [18], а G/LabView дає гірші результати. Тому підходи-кандидати використовують Simscape для виведення рівнянь зі схем або обчислення рівнянь, або вручну, або починаючи з моделей MATLAB чи Simulink.

Схеми buck і full-bridge були змодельовані в HDL за допомогою Simscape, з прийнятною максимальною швидкістю, але набагато вищим використанням ресурсів порівняно з іншими підходами. Simscape завжди враховує втрати першого порядку, що загострює проблему використання ресурсів. У таблиці 1.1 наведено результати реалізації, включаючи VHDL та MATLAB код з [17]. Таблиця показує використання таблиць пошуку (LUT – Look-Up Tables), фліп-флопів (FF – Flip-Flops), цифрових сигнальних

процесорів (DSP – Digital Signal Processors) та мінімальний крок моделювання з використанням чисельного методу Ейлера та арифметики з плаваючою комою. Було зроблено висновок, що пряма трансляція зі схеми Simscape завжди призводить до неефективних результатів, як за ресурсами, так і за швидкістю, тоді як трансляція з коду MATLAB також може призвести до такої неефективності. Тому в цій роботі були досліджені можливі причини неефективності.

Таблиця 1.1 – Використання ресурсів та швидкість з моделями з плаваючою комою, згенерованими вручну з VHDL та трансляцією з коду MATLAB, як в [17], так і з Simscape

Схема	Підхід до моделювання	LUTs	FFs	DSPs	Швидкість (нс)
Buck	VHDL	2003	126	9	51
	MATLAB	2575	64	2	72
	Simscape	10244	397	32	100
Full-bridge	VHDL	3646	158	15	70
	MATLAB код	16301	100	7	159
	Simscape	17144	360	32	155

Погані результати підходів C++, High-Level Synthesis та G/LabView, а також відсутність контролю над внутрішньою структурою моделі Simscape та її високе використання ресурсів призвели до вибору високорівневих підходів MATLAB/Simulink для цієї роботи, як детально буде описано в розділі 2.

2 ПРОПОНОВАНИЙ РОБОЧИЙ ПРОЦЕС ПРОЕКТУВАННЯ

У класичному процесі, який призвів до створення HIL-моделі, модель MATLAB або Simulink використовувалася як вхідні дані для інструменту автоматичної генерації коду (рисунок 2.1). Тут пропонуються додаткові проміжні кроки (рисунок 2.2). Ці кроки забезпечать оптимізацію вхідних даних для інструменту автоматичної генерації коду.

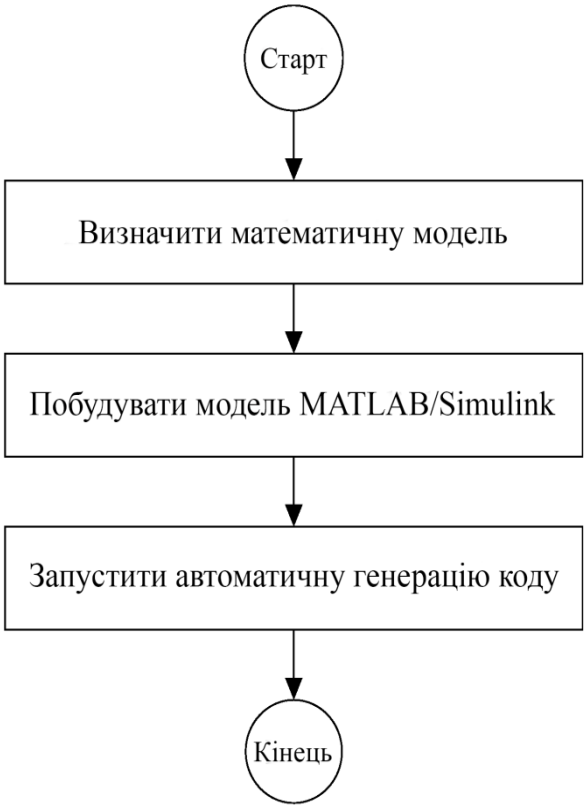

Рисунок 2.1 – Типовий робочий процес проектування для автоматичної генерації коду

Ці кроки будуть добре знайомі досвідченим HDL-дизайнерам. Однак метою цієї роботи було запропонувати метод, який не вимагає знання процесів проектування HDL і є достатньо потужним для забезпечення ефективної HIL-моделі.

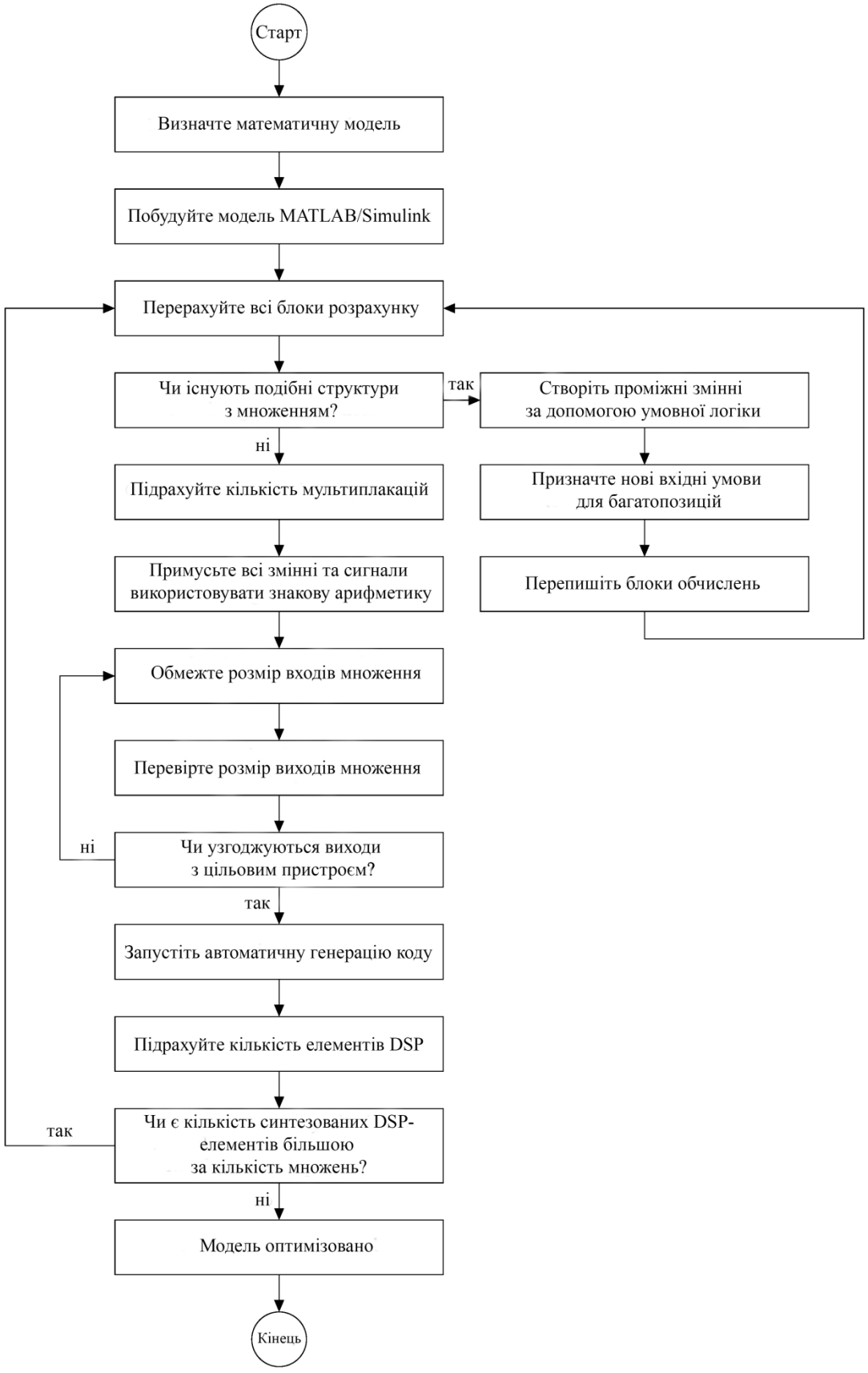

Рисунок 2.2 – Запропонований робочий процес проектування для створення ефективної моделі з автоматичною генерацією коду

Були розглянуті наступні вимоги для запропонованого метода.

1 Метод повинен бути сумісним з існуючими автоматизованими робочими процесами.

2 Він не повинен змінювати початкову математичну модель.

3 Він не повинен вимагати від інженера-енергетика знання мов опису обладнання.

4 Будь-які зміни в структурі алгоритму повинні бути зрозумілими для енергетика.

5 Не вимагати знання внутрішньої структури цільового пристрою.

6 Він не повинен вимагати знання процесів проектування та оптимізації на мові HDL.

7 Він повинен бути достатньо простим, щоб його варто було використовувати, щоб покращення продуктивності кінцевої моделі було отримано ціною невеликих додаткових зусиль.

8 Він повинен бути сумісним з різними чисельними методами.

9 Він повинен бути сумісним зі складними архітектурами, такими як конвеєрна робота.

10 Він повинен досягати аналогічних показників зайнятості в цільовому пристрої в порівнянні з ручним проектуванням.

11 Він повинен досягати такої ж швидкості, як і при ручному проектуванні.

Процес можна розділити на чотири основні етапи: переписування коду, форсування сигналу, адаптація до розміру множника та верифікація.

На етапі переписування коду розробник повинен скласти список усіх обчислювальних блоків, порівняти їх між собою і визначити, чи мають вони схожу структуру. Для тих, що схожі, розробник повинен створити проміжні змінні або сигнали, які приймають своє значення залежно від певних умов. Мета полягає в тому, щоб замінити кілька множень одним, причому всі входи можуть бути активовані або не активовані залежно від умов моделі. Після виконання цієї оптимізації розробник повинен підрахувати кількість

множень. Це буде використано на наступному етапі.

Після цього всі змінні та сигнали примусово переводяться у знакову арифметику, щоб уникнути неконтрольованого додавання бітів. Потім, на третьому етапі, розміри змінних або сигналів, що використовуються при множенні, повинні бути перераховані та обрізані для пристосування до елементів множення пристрою. Це єдиний крок, на якому потрібні певні знання про цільовий пристрій, а саме розміри вхідних і вихідних сигналів елементів множення. Однак ця інформація міститься в будь-якому даташиті і не вимагає глибоких знань.

Потім необхідно перевірити розмір вихідних сигналів множення. Якщо вони більші за розмір пристрою, попередній крок слід повторювати ітеративно, доки виходи не будуть відповідати виходам множника пристрою.

Додатковий крок слугує для перевірки того, що в результаті трансляції отримано оптимізовану модель. Це досягається шляхом порівняння кількості множників, необхідних цільовому пристрою, з кількістю множень, підрахованих після оптимізації обчислювальних структур. У випадку з пристроями Xilinx множниками є DSP-структури. Якщо кількість елементів множника дорівнює кількості множень, це означає, що автоматична генерація коду не створила накладних витрат і модель оптимізована. Тоді можна запускати автоматичну генерацію коду.

Представлений робочий процес відповідає Вимогам 1-6, зазначеним вище. Модифікації прості у виконанні і виконуються на тій самій моделі MATLAB або Simulink, створеній дизайнером, а не на рівні HDL. Вони не змінюють модель і вимагають лише знання елементів множника цільового пристрою. Решта вимог були підтверджені результатами, наведеними в подальших розділах.

3 ХАРАКТЕРИСТИКА В BUCK-ПЕРЕТВОРЮВАЧІ

Початкове дослідження було виконано за допомогою синхронного широтно-імпульсного перетворювача, показаного на рисунку 3.1. Топологія достатньо проста, щоб дозволити ручну перевірку та порівняння HDL-коду, згенерованого різними підходами. Після того, як методологія була проаналізована і зрозуміла, її було застосовано до більш складного перетворювача в розділі 4.

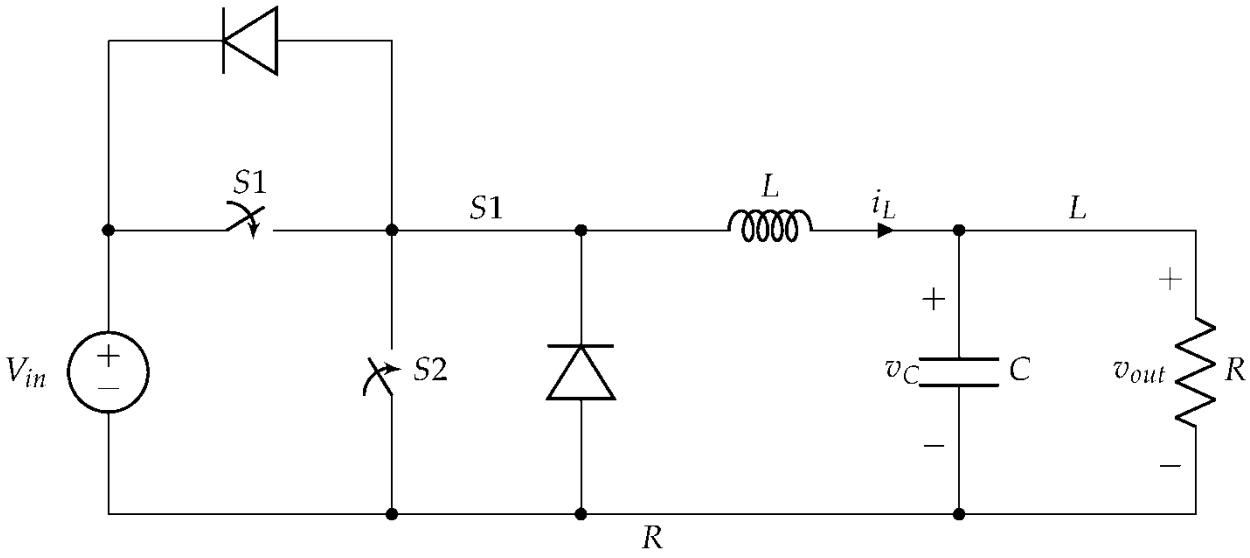

Рисунок 3.1 – Обрана синхронна схема з buck-автоматом

Синхронний перетворювач працює шляхом почергового замикання вимикачів S1 і S2, при цьому робочий цикл визначається співвідношенням v_{out}/v_{in}. Щоб уникнути короткого замикання джерела, між замиканням одного вимикача і розмиканням іншого було введено мертвий час, під час якого діоди D1 і D2 забезпечують шлях для розрядки індуктора.

У схемі можна виділити три режими роботи:

- режим 1: S1 замкнений, а S2 розімкнений або S1 і S2 розімкнені (мертвий час), а струм котушки індуктивності (i_L) від'ємний (D1 у стані

провідності);

- режим 2: S1 розімкнутий, а S2 замкнутий і під час неробочого ходу, в якому i_L>0 (D2 у стані провідності);

- режим 3: якщо котушка індуктивності розряджається під час неробочого ходу, струм через неї не протікає.

Рівняння схеми були отримані на основі поведінки котушки індуктивності та конденсатора, а також режимів роботи перетворювача. Розв'язок для змінних стану v_C та i_L було отримано три набори звичайних диференціальних рівнянь. У таблиці 3.1 наведено рівняння, які застосовуються до кожного випадку.

Таблиця 3.1 – Рівняння buck-схеми для кожного режиму роботи

Режим 1	Режим 2	Режим 3
$\frac{dv_C}{dt} = \frac{i_L}{C} - \frac{v_C}{R \times C}$ $\frac{di_L}{dt} = \frac{v_{in}}{L} - \frac{v_C}{L}$	$\frac{dv_C}{dt} = \frac{i_L}{C} - \frac{v_C}{R \times C}$ $\frac{di_L}{dt} = -\frac{v_C}{L}$	$\frac{dv_C}{dt} = -\frac{v_C}{R \times C}$ $\frac{di_L}{dt} = 0$

Щоб уникнути прорахунків під час переходу з режиму 1 або 2 на режим 3 під час неробочого ходу, в роботі [19] запропоновано різні методи, зокрема насичення струму індуктора до нуля ампер. Це просте і ефективне рішення при використанні методу прямого Ейлера, як в даному випадку.

Модель розраховує в реальному часі розв'язок цих рівнянь. Метод Ейлера широко використовується, оскільки його реалізація є простою. Такі методи, як метод Рунге-Кутта другого або четвертого порядку є більш точними, але складнішими [4,9]. Метод Ейлера з фіксованим кроком дає наступне наближення для малих значень Δt:

$$\frac{dv_C}{dt} = \frac{\Delta v_C}{\Delta t}, \tag{3.1}$$

$$\frac{di_L}{dt} = \frac{\Delta i_L}{\Delta t}. \quad (3.2)$$

Підставляючи (3.1) і (3.2) для кожного режиму, зміни змінних стану i_L і v_C протягом одного кроку dt обчислюються чисельно за допомогою псевдокоду, показаного на рисунку 3.2.

Параметри $1/R$, dt/C та dt/L є константами. Поділ виконується користувачем лише один раз перед початком моделювання. Цей псевдокод буде основою для кожного з проаналізованих розрахункових потоків. Параметри запропонованого бакперетворювача наведено в таблиці 3.2.

Таблиця 3.2 – Параметри запропонованого buck-перетворювача

v_{in}	v_{out}	C	L	P	f_{sw}	dt
25 В	10 В	35 мк Ф	850 мкГн	3,5 Вт	10 кГц	1 мкс

```
1:  Constant declaration
2:  dtC ← dt/C
3:  dtL ← dt/L
4:  invR ← 1/R
5:  function CALCULATESTEP(iL, vC, dtC, dtL, invR, S1, S2, v_in)
6:      if S1 = closed OR (S1 = open AND S2 = open AND iL < 0) then        ▷ Mode 1
7:          ΔvC ← (iL − vC × invR) × dtC
8:          ΔiL ← (v_in − vC) × dtL
9:      else if S2 = closed OR (S1 = open AND S2 = open AND iL > 0) then   ▷ Mode 2
10:         ΔvC ← (iL − vC × invR) × dtC
11:         ΔiL ← −vC × dtL
12:     else if iL = 0 then                ▷ Mode 3: iL = 0 during a dead time
13:         ΔvC ← −vC × invR × dtC
14:         ΔiL ← 0
15:     end if
16:     vC_next ← vC + Δv_C
17:     iL_next ← iL + Δi_L
18:     if sign(iL_next) ≠ sign(iL) AND S1 = open AND S2 = open then   ▷ iL crosses zero during a dead time
19:         iL_next ← 0
20:     end if
21: end function
```

Рисунок 3.2 – Розрахунок наступного значення v_C та i_L методом Ейлера з використанням простого методу насичення iL

3.1 Робочі процеси проектування HDL-моделей

Три робочі процеси HIL починаються з однієї моделі (рисунок 3.2) і мають різну початкову реалізацію: код MATLAB, блочне проектування Simulink і ручний код VHDL. Ця реалізація була змодельована з плаваючою комою. Потім вона була перетворена у фіксовану точку для оптимізації продуктивності. Хоча комерційні HIL-системи зазвичай використовують плаваючу комірку, і деякі спеціальні HIL-системи також використовують її [20,21], більшість спеціальних систем використовують фіксовану комірку.

3.1.1 Перетворення коду MATLAB у VHDL

Функція MATLAB реалізує рівняння стану та обчислює значення змінних стану v_C та i_L по одному кроку за раз, при цьому постійні змінні зберігають свої значення. Інша функція слугувала тестовим стендом, забезпечуючи стимули та записуючи вихідні дані. Потім код перетворюється у версію з фіксованою комою за допомогою перетворення з фіксованою комою робочого процесу генерації коду HDL, причому користувач може регулювати розмір до досягнення необхідної точності. Потім HDL Coder перетворює код з фіксованою комою в код VHDL, який можна синтезувати. Початкова функція MATLAB нагадує алгоритм з рисунку 3.2, з блоком if-then-else, який обчислює Δv_C і Δi_L, визначаючи, чи i_L перетинає нуль під час мертвого часу і встановлює його в нуль, якщо так, то зберігає змінні для наступного циклу обчислень.

3.1.2 Проектування на основі моделей у Simulink

Simulink – це середовище для моделювання та імітаційного моделювання для інженерії на основі моделей. Воно дозволяє здійснювати графічне проектування систем шляхом з'єднання блоків. Для забезпечення

трансляції блоків у мову HDL необхідно використовувати бібліотеку HDL Coder Simulink. Simulink-модель, показана на рисунку 3.2, складається з чотирьох підсистем: селектора режимів, блоків обчислення Ейлера для v_C і i_L та детектора насичення i_L під час неробочого ходу. Параметри $1/R$, dt/C та dt/L є вхідними константами, а стан перемикачів $S1$ та $S2$ і вхідна напруга vin є вхідними сигналами. Блок вибору режиму приймає на входи стани перемикачів $S1$, $S2$ та iL і виробляє вихід (1, 2, 3), який представляє режим роботи. Цей вихід використовується як вхід для інших обчислювальних блоків.

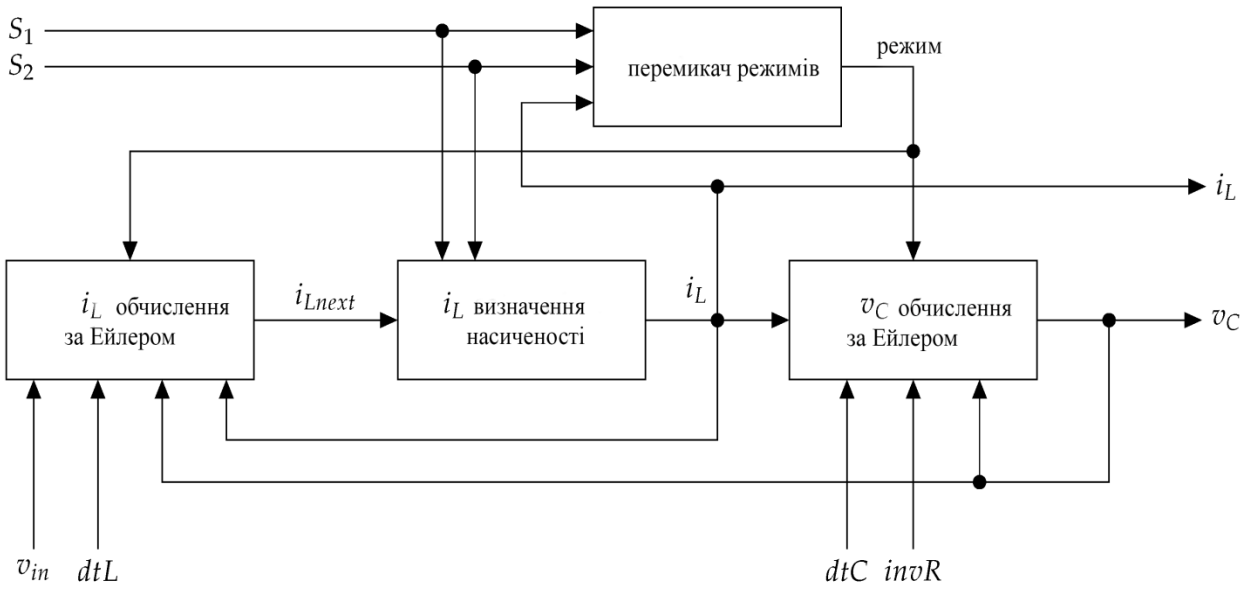

Рисунок 3.3 – Огляд Simulink-моделі buck-конвертера

Блок, який обчислює v_C методом Ейлера (рисунок 3.4), складається з елемента, який обчислює Δv_C, відповідно до режиму роботи, який додається до значення v_C, отриманого через ланцюг зворотного зв'язку через одиничну затримку. У цьому випадку функція f обчислює Δv_C наступним чином: $f(v_C,i_L,...)=(i_{L2}\text{-}v_{Cn\text{-}1}\times invR)\times dtC$, де i_{L2} приймає значення $i_{Ln\text{-}1}$ або нуль залежно від сигналу режиму. Блок розв'язувача Ейлера, доступний у Simulink, не було використано, оскільки він не дозволяв легко призначати dt як зовнішній

сигнал. Діючи, як описано вище, dt можна задати ззовні, присвоївши відповідні значення сигналам dtC та dtL.

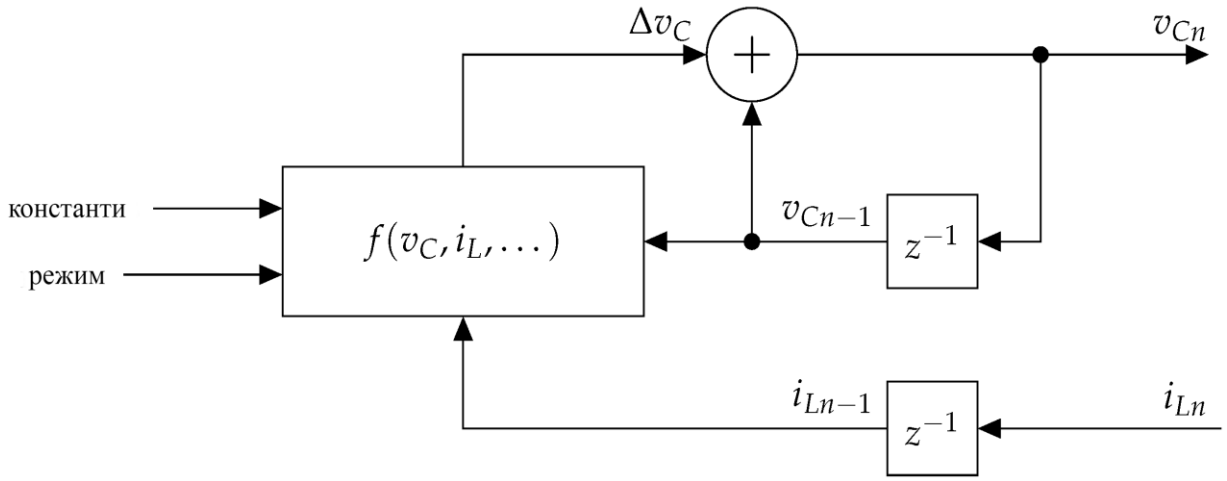

Рисунок 3.4 – Крупний план блоку обчислення vC за Ейлером на рисунку 3.3

Подібна структура використовується для обчислення проміжного значення струму індуктора, i_{Lnext}. Цей сигнал потім подається на блок детектора насичення i_L. Детектор перевіряє, чи i_{Lnext} змінює знак протягом мертвого часу, і якщо так, він встановлює вихід i_L на нуль, доки один з перемикачів не буде знову активовано. В іншому випадку вихід зберігає значення i_{Lnext}.

Кроки для отримання HDL-коду подібні до кроків у підході MATLAB з розділі 3.1.1. Модель Simulink конвертується у формат з фіксованою комою, а HDL Coder автоматично генерує HDL код.

3.1.3 Проектування на основі моделей у Simulink

Для коду на VHDL було застосовано підхід, запропонований у [22]. Ця процедура дала результати, подібні до перекладу з допомогою інструментів MATLAB, описаного в розділах 3.1.1 і 3.1.2, але вимагала більше ручної роботи. Для об'єктивного порівняння було використано уніфікований набір

розмірів (таблиця 3.3) для трьох робочих процесів.

Таблиця 3.3 – Розмір змінних та констант, що використовуються для buck-моделі

Змінна чи постійна	Знак	Цілі біти	Дробові біти	Сумарні біти
i_L	так	7	24	32
i_{Lnext}	так	7	24	32
v_C	так	10	21	32
Δi_L	так	–5	36	32
Δv_C	так	–3	33	32
invR	ні	–4	36	32
dtC	ні	–5	37	32
dtL	ні	–9	41	32
Vin	ні	5	27	32

3.2 Реалізація

3.2.1 Генерація VHDL коду з MATLAB

Відповідно до процесу, описаного в розділі 3.1.1, код MATLAB, еквівалентний алгоритму з рисунку 3.2, було переведено у код з фіксованою комою, а потім у код VHDL для ПЛІС – у даному випадку Xilinx Zynq xc7z020clg400-1. Це недорога ПЛІС, яка має вбудований процесор Advanced RISC Machine (ARM), хоча в моделі він не використовується. Розміри змінних та констант наведено в таблиці 3.3. Від'ємні значення для цілих чисел означають, що крайній лівий біт, який зберігається, знаходиться

праворуч від десяткової крапки. Перший рядок у таблиці 3.4 містить результати реалізації з використанням Xilinx Vivado.

Таблиця 3.4 – Використання ресурсів і максимальна швидкість для buck-перетворювача

Тип коду	LUTs	FFs	DSPs	Швидкість (нс)
VHDL з MATLAB	521	68	20	24
VHDL з Simulink	363	64	12	23

3.2.2 Генерація VHDL коду з Simulink

Було дотримано запропонованого робочого процесу для Simulink-моделі, описаного в розділі 3.1.2, і створено VHDL-код. Розміри сигналів також відповідають таблиці 3.3. Другий рядок у таблиці 3.4 показує результати реалізації. Максимальна швидкість була однаковою в обох випадках; код, створений у MATLAB, використовував більше DSP.

3.2.3 Первісне порівняння

Максимальна швидкість була схожою, але спостерігалася значна різниця у використанні ресурсів, особливо DSP. На використання ресурсів впливали розміри змінних та кількість операцій у ПЛІС; операції множення зазвичай покладаються на фрагменти DSP. В обох випадках було проаналізовано VHDL-код для визначення кількості реалізованих операцій та їх походження з високорівневої моделі.

У першому випадку (код, згенерований в MATLAB) архітектура VHDL була еквівалентна оригінальному високорівневому коду, тоді як у другому випадку (код, згенерований в Simulink) кожен блок в моделі був переведений в код VHDL. Пряме порівняння між двома типами VHDL-коду неможливе,

але існує кореляція між кількістю арифметичних операцій та використанням DSP. У таблиці 3.5 наведено порівняння кількості операцій у кожній моделі та отриманого VHDL-коду.

Таблиця 3.5 – Використання ресурсів і максимальна швидкість для buck-перетворювача

Тип коду	Додавання і віднімання	Множення
MATLAB код	5	6
VHDL з MATLAB	5	6
Simulink схема	4	3
VHDL з Simulink	4	3

3.2.4 Переглянута алгоритмічна модель

Для оптимізації алгоритму 1 з рисунку 3.2 слід уникати алгебраїчних операцій. Один із способів – обмежити блок if-then-else лише присвоєннями, з режимом роботи схеми (1, 2 або 3) на виході. Тоді проміжні змінні використовуються для зберігання значень, які залежать від режиму роботи, а алгебраїчні операції, які обчислюють Δi_L і Δv_C, повинні з'являтися лише один раз. Алгоритм 2 ілюструє цей оптимізований підхід.

Ця альтернатива зробила архітектуру схожою на підхід на основі Simulink: блок if-then-else нагадує блок вибору режиму, вихід якого керує перемиканням сигналів, які, в свою чергу, використовуються для обчислення Δi_L і Δv_C.

Переглянутий псевдокод містить лише три операції множення, чотири операції додавання або віднімання та одну операцію зміни знаку. Як результат, він генерує оптимальний VHDL-код, що дає подібні результати по зайнятості та швидкості, як і підхід на основі Simulink, як показано в таблиці 3.6. Хоча різниця в максимально досяжній швидкості була незначною, накладні витрати були зумовлені генерацією додаткових сигналів у

паралельних шляхах обчислень, що спричинило додаткове використання ресурсів. Однак це не впливало на час, оскільки логіка виконувала ці обчислення паралельно, а шлях обчислення мав однакову довжину.

Constant declaration
$dtC \leftarrow dt/C$
$dtL \leftarrow dt/L$
$invR \leftarrow 1/R$
function CALCULATESTEP($iL, vC, dtC, dtL, invR, S1, S2, v_{in}$)
    $iC_{temp} \leftarrow vC \times invR$
    if $S1 = closed$ OR ($S1 = open$ AND $S2 = open$ AND $iL < 0$) **then**
        $mode \leftarrow 1$
    else if $S2 = closed$ OR ($S1 = open$ AND $S2 = open$ AND $iL > 0$) **then**
        $mode \leftarrow 2$
    else if $iL = 0$ **then**
        $mode \leftarrow 3$
    end if
    if $mode = 1$ OR $mode = 2$ **then**
        $iL_{temp} \leftarrow iL$
    else
        $iL_{temp} \leftarrow 0$
    end if
    $\Delta vC \leftarrow (iL_{temp} - iC_{temp}) \times dtC$
    if $mode = 1$ **then**
        $v2_{temp} \leftarrow v_{in} - vC$
    else if $mode = 2$ **then**
        $v2_{temp} \leftarrow -vC$
    else
        $v2_{temp} \leftarrow 0$
    end if
    $\Delta iL \leftarrow v2_{temp} \times dtL$
    $vC_{next} \leftarrow vC + \Delta v_C$
    $iL_{next} \leftarrow iL + \Delta i_L$
    if $sign(iL_{next}) \neq sign(iL)$ AND $S1 = open$ AND $S2 = open$ **then** ▷ iL crosses zero during a dead time
        $iL_{next} \leftarrow 0$
    end if
end function

Рисунок 3.5 – Оптимізована версія алгоритму 1, яка вилучає множення з блоків if-then

3.2.5 Переглянута алгоритмічна модель

Реалізація на мові VHDL відповідає структурі, представленій у розділі 3.1.3, з урахуванням розмірів сигналів, наведених у таблиці 3.3. Результати наведено в четвертому рядку таблиці 3.6. Цей робочий процес генерує найменше використання ресурсів, оскільки VHDL-проектувальник використовує лише те, що необхідно для безпосередньої реалізації алгоритму

на мові низького рівня. Незважаючи на це, максимальна досягнута швидкість була подібною до інших методів, що свідчить про те, що автоматичні інструменти можуть транслювати арифметичні операції у високопродуктивний HDL-код з ефективним обчислювальним трактом.

Таблиця 3.6 – Використання ресурсів і максимальна швидкість для buck-моделі

Тип коду	LUTs	FFs	DSPs	Швидкість (нс)
VHDL з MATLAB	521	68	20	24
VHDL з Simulink	363	64	12	23
VHDL з оновленого MATLAB	400	67	12	23
Ручний VHDL	287	64	9	25

3.3 Метрики коду та використання ресурсів

Метрики можуть бути використані для порівняння складності VHDL коду та накладних витрат, що генеруються автоматичним перекладом. Кількість рядків коду часто використовується в програмному забезпеченні [23] і може бути застосована до VHDL-коду. Це дає уявлення про накладні витрати, пов'язані з перекладом, але ці витрати не обов'язково призводять до додаткового використання логіки. Інші пропозиції (додавання/віднімання, множення, блоки if-then-else, кількість сигналів та змінних) мають прямий зв'язок з використанням ресурсів. Вище число в будь-якому з них означає більше використання ресурсів. Ці метрики, наведені в таблиці 3.7, дозволяють провести детальний аналіз складності VHDL коду.

Ці показники коду самі по собі не пояснюють, чому моделі MATLAB і Simulink використовують більше ресурсів. Можна припустити, що кількість сигналів і змінних корелює з більшим використанням LUT і FF. Однак у коді

є три множення, і вони перетворюються на 12 DSP замість 9, як у випадку ручного VHDL-коду.

Перевірка згенерованого коду показала, що сигнали зростали в проміжних обчисленнях, а входи до множників не були оптимізовані. Беззнакові сигнали можуть зростати на один біт, якщо вони втручаються в обчислення зі знаковими сигналами. Таким чином, за допомогою автоматизованих інструментів користувач не може визначити точний розмір входів множників, якщо не створити проміжну змінну або не розмістити блок перетворення перед множенням.

Таблиця 3.7 – Метрики VHDL коду трьох потоків проектування

Характеристика	MATLAB (початковий)	MATLAB (оптимізований)	Simulink	Ручний VHDL
Рядки коду	306	284	1007	107
Додавання/віднімання	5	4	4	4
Множення	6	3	3	3
Блоки If-then-else	13	13	14	5
Сигнали та змінні	102	89	95	13
Процеси	6	6	7	2

Вбудовані блоки DSP у цільовій ПЛІС включають перемножувачі з одним входом на 25 біт і одним на 18 біт. Обидва входи є знаковими. Якщо в будь-якому множенні використовується більше бітів, інструмент синтезу буде використовувати два або більше DSP-блоків для одного множення. Це не тільки збільшить необхідні ресурси (блоки DSP), але й затримку, оскільки вона включатиме затримку обох блоків DSP.

4 ВАЛІДАЦІЯ В БІЛЬШ СКЛАДНІЙ СХЕМІ

У цьому розділі представлено більш складну модель для перевірки запропонованого робочого процесу проектування. Зокрема, складніша модель була використана як для перевірки вимог з 7 по 11, так і для перевірки того, чи достатньо простого дотримання запропонованого робочого процесу з рисунка 2.2 для отримання HIL-моделі з характеристиками, подібними до характеристик моделі, створеної вручну, тобто чи достатньо запропонованого робочого процесу для забезпечення оптимізованої моделі в реальному часі без знання HDL-проектування. Було обрано повномостовий перетворювач з електричними втратами першого порядку, показаний на рисунку 4.1, з використанням методу Рунге-Кутта другого порядку. Цей метод забезпечує високу точність навіть при великих кроках моделювання. Він вимагає двох обчислень під час кожного обчислювального циклу замість одного, тому є кандидатом на конвеєрну реалізацію. Оскільки він є більш складним, автоматизований робочий процес, безумовно, був би кращим.

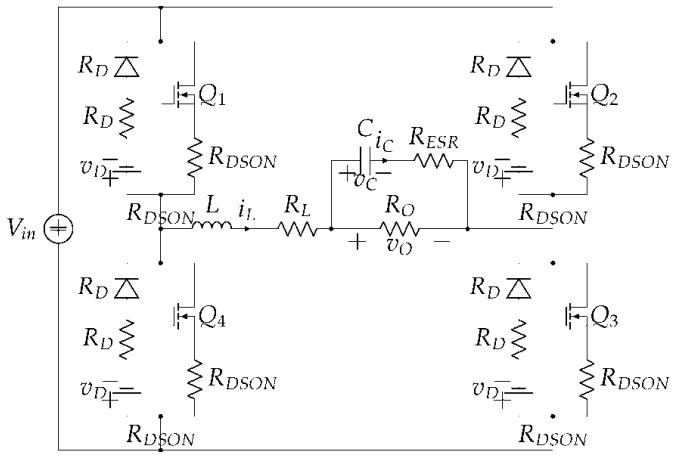

Рисунок 4.1 – Повномостовий перетворювач з втратами, обраний для перевірки запропонованого робочого процесу

4.1 Рівняння повномостового перетворювача

Початкові рівняння, що описують поведінку схеми [13], були переписані для застосування методу Рунге-Кутта другого порядку:

$$\frac{di_L}{dt} = M_1 \times i_L + M_2 \times v_C + M_3, \tag{4.1}$$

$$\frac{dv_C}{dt} = N_1 \times i_L + N_2 \times v_C. \tag{4.2}$$

Значення для M_i та N_i мають вигляд:

$$M_1 = \frac{-R_{ESR}}{L\times(1+G_O\times R_{ESR})} - \frac{KR}{L}, \tag{4.3}$$

$$M_2 = \frac{R_{ESR}\times G_O}{L\times(1+G_O\times R_{ESR})} - \frac{1}{L}, \tag{4.4}$$

$$M_3 = \frac{-KV_1\times V_{in} - KV_2}{L}, \tag{4.5}$$

$$N_1 = \frac{1}{C\times(1+G_O\times R_{ESR})}, \tag{4.6}$$

$$N_2 = -\frac{G_O}{C\times(1+G_O\times R_{ESR})}. \tag{4.7}$$

Терміни R_L, R_D, R_{DSON} та R_{ESR} є послідовним опором котушки індуктивності, діода, польового транзистора на основі металу-оксиду-напівпровідника (MOSFET) та конденсатора відповідно, а G_O=1/R_O. Пряма напруга діода дорівнює V_D. Величини KR, KV_1 та KV_2 змінюються в залежності від умов роботи схеми. Їх значення визначаються станом

перемикачів Q_1 до Q_4 та знаком струму котушки індуктивності i_L згідно з таблицею 4.1. Решта членів залишаються незмінними під час роботи схеми. Рівняння для v_O можна записати наступним чином:

$$v_O = P_1 \times i_L + P_2 \times v_C, \tag{4.8}$$

$$P_1 = \frac{R_{ESR}}{1+G_O \times R_{ESR}}, \tag{4.9}$$

Таблиця 4.1 – Параметри KR, KV_1 та KV_2 відповідно до стану перемикачів та знаку струму індуктивності

Q_1, Q_2, Q_3, Q_4	i_L	KR	KV_1	KV_2
ON, OFF, ON, OFF	будь-яке	V_{in}	0	$2R_{DSON} + R_L$
OFF, ON, OFF, ON	будь-яке	$-V_{in}$	0	$2R_{DSON} + R_L$
OFF, OFF, OFF, OFF	>0	$-V_{in}$	$2V_D$	$2R_D + R_L$
OFF, OFF, OFF, OFF	>0	V_{in}	$-2V_D$	$2R_D + R_L$
ON, OFF, OFF, OFF	>0	0	V_D	$R_{DSON} + R_D + R_L$
OFF, OFF, ON, OFF	>0	0	V_D	$R_{DSON} + R_D + R_L$
ON, OFF, OFF, OFF	>0	V_{in}	$-V_D$	$R_{DSON} + R_D + R_L$
OFF, OFF, ON, OFF	>0	V_{in}	$-V_D$	$R_{DSON} + R_D + R_L$
OFF, ON, OFF, OFF	>0	V_{in}	V_D	$R_{DSON} + R_D + R_L$
OFF, OFF, OFF, ON	>0	V_{in}	V_D	$R_{DSON} + R_D + R_L$
OFF, ON, OFF, OFF	>0	0	$-V_D$	$R_{DSON} + R_D + R_L$
OFF, OFF, OFF, ON	>0	0	$-V_D$	$R_{DSON} + R_D + R_L$

$$P_2 = 1 - \frac{R_{ESR} \times G_O}{1 + G_O \times R_{ESR}}. \quad (4.10)$$

4.2 Модель Рунге-Кутта другого порядку з К-калькулятором

Метод Рунге-Кутта другого порядку використовує двокрокове наближення для обчислення змінних стану. Для запропонованого повного моста члени обчислюються наступним чином:

$$K_{1L} = M_1 \times i_L + M_2 \times v_C + M_3, \quad (4.11)$$

$$K_{1C} = N_1 \times i_L + N_2 \times v_C, \quad (4.12)$$

$$K_{2L} = M_1 \times (i_L + dt \times K_{1L}) + M_2 \times (v_C + dt \times K_{1C}) + M_3, \quad (4.13)$$

$$K_{2C} = N_1 \times (i_L + dt \times K_{1L}) + N_2 \times (v_C + dt \times K_{1C}). \quad (4.14)$$

Значення змінних стану на наступному часовому кроці задаються за допомогою :

$$i_{L_n+1} = i_{L_n} + dt \times (K_{1C} + K_{2C})/2, \quad (4.14)$$

$$v_{C_n+1} = v_{C_n} + dt \times (K_{1C} + K_{2C})/2. \quad (4.15)$$

Реалізація повного мосту використовує К-калькулятор, подібний до запропонованого в [19]. К-калькулятор – це обчислювальний блок, який обчислює різні K_i члени методу Рунге-Кутта – K_1 коли вхідні дані дорівнюють нулю та K_2 коли на вході є члени K_1 члени. Це дозволяє обчислювати ці доданки ітеративно, повторно використовуючи апаратні

елементи. К-обчислювач реалізує ці рівняння:

$$K_{outL} = M_1 \times (i_L + dt \times K_{inL}) + M_2 \times (v_C + dt \times K_{inC}) + M_3, \quad (4.16)$$

$$K_{outC} = N_1 \times (i_L + dt \times K_{inL}) + N_2 \times (v_C + dt \times K_{inC}). \quad (4.17)$$

4.3 Повномостова модель MATLAB

Код MATLAB базувався на початковому коді методу прямого Ейлера, використаному в [13], з перенесенням множень з блоків if-then, щоб уникнути апаратного дублювання. Процедурний код з двома викликами функції К-калькулятора було переписано для реалізації конвеєрної операції з машиною станів:

- стан 0: обчислює M_1 та M_3;

- стан 1: К-калькулятор обчислює K_{1C} та K_{1L} та. Паралельно обчислюється vO обчислюється на основі збережених значень v_C та i_L;

- стан 2: К-обчислювач обчислює K_{2C} та K_{2L};

- стан 3: нові значення iL та vC обчислено.

Функція К-калькулятора викликається один раз поза блоками if-then для повторного апаратного використання. Стани 1 і 2 мають максимальну латентність через два послідовних множення. Перетворення з фіксованою комою виконувалося за допомогою радника HDL Code Generation Workflow. Розмір вхідного сигналу було обмежено 25 або 18 бітами зі знаковим бітом, щоб уникнути додаткового синтезу DSP, як запропоновано в розділі 2.

4.4 Повномостова симуляційна модель у Simulink

Модель Simulink використовує конвеєрну конфігурацію з чотирма станами, подібно до моделі MATLAB. На рисунку 4.2 показано спрощену

версію моделі з чотирма блоками, по одному для кожного стану. Вони виконують обчислення M_1 та M_3, K-обчислення, накопичення параметрів K_i та присвоєння значень змінним стану i_L та v_C. Блоки перетворення використовуються на входах для забезпечення належних розмірів змінних та адаптації змінних перед множенням. Наприклад, i_L та v_C перед множенням адаптуються до 25 біт для обчислення v_O. Це забезпечує мінімальну кількість DSP.

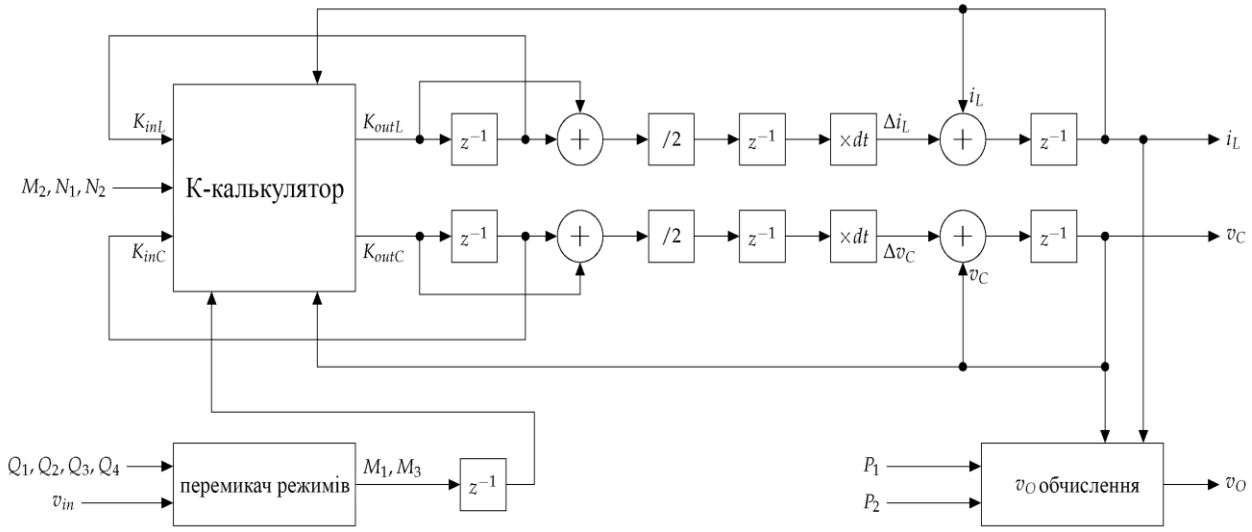

Рисунок 4.2 – Огляд Simulink-моделі повномостового випрямляча з втратами

4.5 Повномостова нативна модель VHDL

Нативна VHDL-модель повного мостового перетворювача містить одну процедуру, два процеси та K-обчислювач у комбінаторній логіці. Процедура обчислює змінні M_1 та M_3. Перший процес містить операції автомата. Він викликає процедуру для обчислення M_1 та M_3 у першому стані кожної ітерації. Він встановлює правильні вхідні дані для K-обчислювача (або нуль, або значення K_1 з першого циклу) і обчислює оновлення змінних стану в останньому стані ітерації. Другий процес оновлює зареєстровані виходи.

4.6 Підсумок результатів повномостової моделі

Три підходи показали схожі результати з точки зору максимальної швидкості та використання пристроїв, причому всі вони потребують 12 DSP. Що стосується LUT, то VHDL показав значно кращі результати. Максимальна тактова частота коливалася від 19 до 21,2 нс. Використовуючи чотириступеневий конвеєр, результат у реальному часі за допомогою методу Рунге-Кутта видавався б кожні 80 нс, що приблизно в п'ять разів повільніше, ніж початковий метод Ейлера. Однак, оскільки результати були більш точними, загальна продуктивність системи була б кращою. У таблиці 4.2 наведено результати трьох підходів і порівняння з методом, запропонованим у [17]. Модель вимагала більше ресурсів, оскільки чисельний метод був складнішим, але різниця між автоматичним і ручним дизайном була незначною, на відміну від двох порівнянних моделей, створених у [17].

Таблиця 4.2 – Використання ресурсів та максимальна швидкість для моделі з повним мостом (рядки 1-3) та порівняння з методом, запропонованим в [17] (рядки 4 і 5)

Тип коду	LUTs	FFs	DSPs	Швидкість (нс)	Чисельний метод
VHDL з MATLAB	603	299	12	21,2	Рунге-Кутта другого порядку
VHDL з Simulink	602	513	12	19,0	Рунге-Кутта другого порядку
Ручний VHDL	397	326	12	21,0	Рунге-Кутта другого порядку
VHDL з MATLAB	837	97	10	19,8	Ейлер
Ручний VHDL	759	77	7	19,3	Ейлер

У таблиці 4.3 порівнюються метрики трьох потоків проектування моделі повного моста. Напівавтоматично згенерований VHDL-код був

складнішим. Однак параметри, що безпосередньо впливають на зайнятість апаратного забезпечення та продуктивність, були подібними до параметрів коду VHDL, написаного вручну. Ручний VHDL-код мав 14 множень замість 12, але в кінцевому підсумку використовував 12 DSP. Це сталося тому, що K-калькулятор у VHDL-коді був написаний відповідно до рівнянь (4.16) і (4.17), що дозволило інструменту синтезу згенерувати правильну оптимізацію і створити лише 2 DSP для 4 доданків множення.

Ці результати показали, що запропонований робочий процес ефективно забезпечує перетворення в ефективну модель. Навіть при обробці більш складної моделі, яка зазвичай розробляється фахівцями з HDL, ітеративні оптимізації коду MATLAB і Simulink призвели до створення дуже ефективних HDL-моделей, повністю порівнянних з ручною моделлю з точки зору швидкості та використання ресурсів. Це примітно, оскільки ручна модель генерувалася з більшими зусиллями і вимагала знання мови VHDL. У випадку з моделями MATLAB і Simulink цього не потрібно було робити.

Таблиця 4.3 – Порівняння метрик для моделі повного мосту з втратами

Характеристика	MATLAB	Simulink	Ручний VHDL
Рядки коду	928	1288	147
Додавання/віднімання	13	13	14
Множення	12	12	14
Блоки If-then-else	22	59	5
Сигнали та змінні	338	230	16
Процеси	12	29	2

5 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

Останнім кроком є апаратна реалізація системи. Як пояснювалося в розділі 3.2.1, було використано пристрій Zynq. У реалізованому проекті програмована логіка використовувалася для запуску моделі в реальному часі, а процесор зв'язувався із зовнішнім комп'ютером для конфігурації.

Блок-схему Simulink (розділ 4.4) було модифіковано шляхом додавання портів Advanced eXtensible Interface (AXI) для початкової конфігурації, а код VHDL було автоматично згенеровано знову для створення ядра інтелектуальної власності (IP). У Vivado було створено блоковий дизайн для інтеграції згенерованого ядра IC в обчислювальну систему Zynq. Модель повномостового перетворювача (параметри показані в таблиці 5.1) працювала в режимі реального часу з тактовою частотою 22 нс. Як видно з таблиці 5.2, використання ПЛІС і тактова частота були вищими за теоретичний мінімум (таблиця 4.2) завдяки додаванню інтерфейсів AXI та інтеграції з процесором.

Таблиця 5.1 – Параметри запропонованого повномостового перетворювача

v_{in}	C	L	R	R_{ESR}	R_D	R_{DSON}	R_L	V_D	f_{sw}	dt
200 В	100 мк Ф	900 мкГн	200 Ом	360 мОм	800 мОм	100 мОм	5 мОм	0,7 В	200 кГц	116 нс

Таблиця 5.2 – Параметри запропонованого повномостового перетворювача

Тип коду	LUTs	FFs	DSPs	Швидкість (нс)
Оголений режим	602	513	12	19
Модель з AXI та мікропроцесором	1567	1773	12	22

Результати були отримані за допомогою інтегрованого логічного аналізатора. На рисунку 5.1 показано перехідний процес від робочого циклу 50% до 75%. Для усунення пульсацій при перемиканні було вилучено лише одну точку за цикл перемикання, хоча менші кроки були розраховані внутрішньо. Модель може розраховувати значення належним чином і може бути використана для проектування та тестування апаратного забезпечення в циклі, як запропоновано в розділі 1.

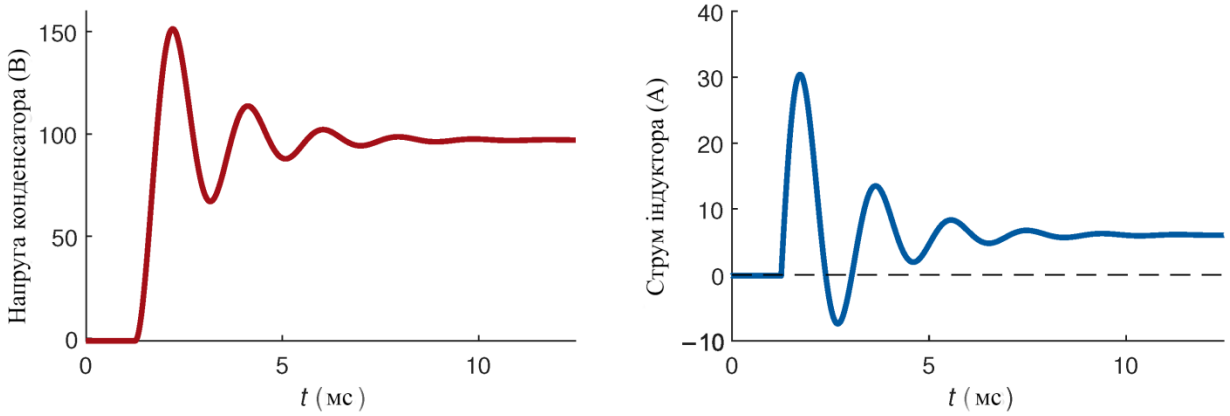

Рисунок 5.1 – Перехід від 50% робочого циклу до 75% робочого циклу розраховується в режимі реального часу

ВИСНОВКИ

Ця робота зосереджена на реалізації моделей з використанням їхніх рівнянь, а не на прямій трансляції схем у код HDL, що може мати низьку продуктивність і високе використання апаратного забезпечення. Результати показали, що напівавтоматично згенеровані моделі досягають високої ефективності, якщо дотримуватися запропонованого методу.

Максимальна швидкість була однаковою у всіх випадках. Використання DSP є критично важливим у HIL-додатках, і неоптимізований алгоритм міг би подвоїти кількість використовуваних DSP, але досяжна швидкість неоптимізованого підходу все одно була подібною до оптимізованого. Найкращі результати показав підхід генерації коду HDL, але він також був найскладнішим і вимагав ручного перетворення у фіксовану крапку. Крім того, включення інтерфейсів AXI вимагало знання протоколів та їхньої реалізації. Нарешті, конвеєрні проекти збільшують складність, підвищуючи ризик неправильної реалізації. Все це робить запропонований робочий процес дуже привабливим для дизайнерів, які не є експертами в HDL, оскільки усуваються зусилля низькорівневих завдань HDL з невеликою доданою вартістю. При цьому дизайнер може бути впевнений, що модель, отримана автоматично, має оптимальну продуктивність. Ціна за це – ретельна робота з моделлю MATLAB або Simulink. Однак така модель добре відома проектувальнику, тому вона вимагає невеликих зусиль і створює низький ризик.

Рекомендований робочий процес виявився сумісним зі складними моделями з конвеєрною архітектурою та кількома чисельними методами. Розробник силової електроніки може створити ефективну модель в реальному часі, внісши прості зміни в реалізацію MATLAB або Simulink. Знання цільового пристрою не є необхідним, за винятком розмірів вхідного та вихідного сигналів помножувача. Модифікації відбуваються в рамках

добре відомої розробнику моделі MATLAB або Simulink: перевірка розрахункових блоків, створення нових сигналів на основі логічних умов, які залежать від добре відомих йому режимів роботи схеми, підстроювання розмірів змінних до тих, що вимагаються пристроєм, і перевірка відповідності кількості елементів помножувача розрахункам в проекті MATLAB/Simulink.

Запропонованих кроків було достатньо для забезпечення оптимальної трансляції коду. Їх можна додати до існуючого процесу проектування, а накладні витрати, пов'язані з додатковою роботою та ітераціями, переважають гарантію того, що буде створено дуже ефективну модель.

Запропоновано модифікацію робочого процесу проектування HIL-моделей, яка призводить до автоматичної генерації дуже ефективних моделей. Це усуває необхідність проектування на рівні HDL і забезпечує результати, дуже близькі до результатів ручного проектування. Він був проаналізований на простому дизайні і підтверджений на більш складному.

Представлена тут пропозиція усуває необхідність вибирати між ефективною моделлю та автоматизованим виробництвом.

Для типового проектувальника, який мало або зовсім не знає мови HDL, застосування цієї пропозиції до підходів на основі MATLAB і Simulink дає змогу отримати ефективний код, з додатковим використанням ресурсів, але з тією ж максимальною швидкістю. Це перспективний шлях, оскільки він дозволяє розробникам схем генерувати моделі з дуже хорошою поведінкою в реальному часі в автоматизованих робочих процесах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mihalič, F.; Truntič, M.; Hren, A. Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges. Electronics 2022, 11, 2462.

2. Kirei, B.S.; Farcas, C.A.; Chira, C.; Ilie, I.A.; Neag, M. Hardware Emulation of Step-Down Converter Power Stages for Digital Control Design. Electronics 2023, 12, 1328.

3. Aiello, G.; Cacciato, M.; Scarcella, G.; Scelba, G. Failure analysis of AC motor drives via FPGA-based hardware-in-the-loop simulations. Electr. Eng. 2017, 99, 1337–1347.

4. Yushkova, M.; Sanchez, A.; de Castro, A. Strategies for choosing an appropriate numerical method for FPGA-based HIL. Int. J. Electr. Power Energy Syst. 2021, 132, 107186.

5. Razzaghi, R.; Mitjans, M.; Rachidi, F.; Paolone, M. An automated FPGA real-time simulator for power electronics and power systems electromagnetic transient applications. Electr. Power Syst. Res. 2016, 141, 147–156.

6. Liu, C.; Ma, R.; Bai, H.; Li, Z.; Gechter, F.; Gao, F. Hybrid modeling of power electronic system for hardware-in-the-loop application. Electr. Power Syst. Res. 2018, 163, 502–512.

7. Ilyashov, O., Pokora, K., Diachenko, V. і Kovalenko, A. 2023. Класифікація даних апаратними прискорювачами FPGA у центрах обробки даних та хмарах. Системи управління, навігації та зв'язку. Збірник наукових праць. 2, 72 (Чер 2023), 106-112. DOI:https://doi.org/https://doi.org/10.26906/SUNZ.2023.2.106.

8. Selvamuthukumaran, R.; Gupta, R. Rapid prototyping of power electronics converters for photovoltaic system application using Xilinx System Generator. IET Power Electron. 2014, 7, 2269–2278.

9. Parizad, A.; Mohamadian, S.; Iranian, M.E.; Guerrero, J.M. Power

System Real-Time Emulation: A Practical Virtual Instrumentation to Complete Electric Power System Modeling. IEEE Trans. Ind. Inform. 2019, 15, 889–900.

10. Siwakoti, Y.P.; Town, G.E. Design of FPGA-controlled power electronics and drives using MATLAB Simulink. In Proceedings of the 2013 IEEE ECCE Asia Downunder, Melbourne, Australia, 3–6 June 2013; pp. 571–577.

11. Alecsa, B.; Cirstea, M.N.; Onea, A. Simulink Modeling and Design of an Efficient Hardware-Constrained FPGA-Based PMSM Speed Controller. IEEE Trans. Ind. Inform. 2012, 8, 554–562.

12. Bonny, T. Chaotic or Hyper-chaotic Oscillator? Numerical Solution, Circuit Design, MATLAB HDL-Coder Implementation, VHDL Code, Security Analysis, and FPGA Realization. Circuits Syst. Signal Process. 2021, 40, 1061–1088.

13. Корнієнко, Є., Ляшенко, О. і Торба, А. (2023) «Метод керування системою генерації електроенергії з використанням бездротових технологій», Сучасний стан наукових досліджень та технологій в промисловості, (2(24), с. 80–89. doi: 10.30837/ITSSI.2023.24.080.

14. Costas, L.; Colodrón, P.; Rodríguez-Andina, J.J.; Fariña, J.; Chow, M.Y. Analysis of two FPGA design methodologies applied to an image processing system. In Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, Bari, Italy, 4–7 July 2010; pp. 3040–3044.

15. Rosado-Muñoz, A.; Bataller-Mompeán, M.; Soria-Olivas, E.; Scarante, C.; Guerrero-Martínez, J.F. FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches. IEEE Trans. Ind. Electron. 2011, 58, 860–870.

16. Karimi, S.; Poure, P.; Saadate, S. An HIL-Based Reconfigurable Platform for Design, Implementation, and Verification of Electrical System Digital Controllers. IEEE Trans. Ind. Electron. 2010, 57, 1226–1236.

17. Barkovska, O., Filippenko, I., Semenenko, I., Korniienko, V., & Sedlaček, P. (2023). Адаптація FPGA архітектури для прискорення алгоритмів обробки зображень. Радіоелектронні і комп'ютерні системи, 0(2), 94-106. doi:https://doi.org/10.32620/reks.2023.2.08.

18. Lamo, P.; Ruiz, G.A.; Azcondo, F.J.; Pigazo, A.; Brañas, C. Impact of the Noise on the Emulated Grid Voltage Signal in Hardware-in-the-Loop Used in Power Converters. Electronics 2023, 12, 787.

19. Saralegui, R.; Sanchez, A.; de Castro, A. Modeling of Deadtime Events in Power Converters with Half-Bridge Modules for a Highly Accurate Hardware-in-the-Loop Fixed Point Implementation in FPGA. Appl. Sci. 2021, 11, 6490.

20. Sanchez, A.; Todorovich, E.; de Castro, A. Impact of the hardened floating-point cores on HIL technology. Electr. Power Syst. Res. 2018, 165, 53–59. [Google Scholar] [CrossRef]

21. Liu, J.; Dinavahi, V. Nonlinear Magnetic Equivalent Circuit-Based Real-Time Sen Transformer Electromagnetic Transient Model on FPGA for HIL Emulation. IEEE Trans. Power Deliv. 2016, 31, 2483–2493.

22. Sanchez, A.; de Castro, A.; Garrido, J. A Comparison of Simulation and Hardware-in-the- Loop Alternatives for Digital Control of Power Converters. IEEE Trans. Ind. Inform. 2012, 8, 491–500.

23. Perez-Cham, O.E.; Montalvo, C.S.; Nunez-Varela, A.S.; Puente, C.; Ontanon-Garcia, L.J. Source Code Metrics to Predict the Properties of FPGA/VHDL-Based Synthesized Products. In Proceedings of the 2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT), San Luis Potosi, Mexico, 24–26 October 2018; pp. 93–98.