Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи функціонального моделювання електронних систем

(тема)

Виконав:
студент II курсу, групи КСМзм-19-1
Бобровник А.А.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма
Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: проф. Горбачов В.О.
(посада, прізвище, ініціали)

Допускається до захисту

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

2020 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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

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

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Бобровнику Анатолію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи функціонального моделювання електронних систем

затверджена наказом по університету від "23" жовтня 2020 р. № 168 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи

Методологія проектування рівня електронної системи

Технології верифікації

Мова VHDL

Методологія проектування рівня електронної системи

4. Перелік питань, що потрібно опрацювати в роботі

Вступ. Аналіз проблеми

Дизайн та перевірка електронних систем

Огляд методології проектування електронного рівня системи

Перевірка функціональності за допомогою аналізу посилань

Транзакційна технологія проектування

Додаток

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 13 арк. ф. А4

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	Аналіз завдання	12.11.2020-16.11.2020	
2	Аналіз науково-технічної літератури	17.11.2020-23.11.2020	
3	Пошук моделі загроз системи	24.11.2020-28.11.2020	
4	Пошук аналітичних моделей комп'ютерних	29.11.2020-07.12.2020	
	систем та мереж		
5	Вивчення концепції імітаційного моделювання	08.12.2020-15.12.2020	
6	Визначення вторгнень	16.12.2020-20.12.2020	
7	Оформлення пояснювальної записки	08.12.2020-11.12.2020	
8	Оформлення графічної частини	08.12.2020-11.12.2020	
9	Представлення магістерської роботи	12.12.2020-13.12.2020	
	науковому керівнику		

Дата видачі завдання 26 жовтня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Горбачов В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 107 с., 38 рис., 2 дод., 16 джерел.

ІНТЕГРОВАНА СХЕМА, РІВЕНЬ ЕЛЕКТРОННОЇ СИСТЕМИ, МОДЕЛЮВАННЯ РІВНЯ ТРАНСАКЦІЇ, РІВЕНЬ РЕГИСТРОВИХ ПЕРЕДАЧ .

У роботі пропонується ефективний метод функціональної перевірки проектів SoC в рамках методології системного рівня.

Упродовж даної атестаційної роботи передбачається досягнення наступних цілей: побудова моделі інтегрованого середовища перевірки на основі ядра TLM, яке забезпечить взаємозв’язок компонентів SoC; створення інтерфейсів взаємозв’язку універсальних компонентів SoC; розробка алгоритму для моделювання аналізу каналів передачі даних; тестування верифікаційного рішення на демонстраційному проекті.

ABSTRACT

Master’s thesis: 107 pages, 38 figures, 2 appendices, 26 sources.

INTEGRATED CIRCUIT, ELECTRONIC SYSTEM LEVEL, TRANSACTION-LEVEL MODELING, REGISTER-TRANSFER LEVEL.

This thesis is to propose effective method for functional verification of SoC projects within system level methodology.

The following goals are supposed to be done throughout of this thesis: constructing model of integrated verification environment on the base of TLM kernel that will provide SoC components interconnection; creating universal SoC components’ interconnection interfaces; developing algorithm for simulation data link analysis; testing the verification solution on demo project.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 9
ВСТУП 10
1 ОГЛЯД ПРОБЛЕМИ ТА МЕТА РОБОТИ 12
1.1 Визначення проблеми 12
1.2 Мета роботи 14
1.3 Огляд літератури 15
1.4 Організація роботи 18
2 ПРОЕКТУВАННЯ ТА ПЕРЕВІРКА ЦИФРОВИХ СИСТЕМ 19
2.1 Вступ до проектування цифрових систем 19
2.2 Сучасні тенденції в дизайні цифрових систем 21
2.2.1 Спільне проектування апаратно-програмного забезпечення 22
2.2.2 Класичні напрямки дизайну 23
2.2.3 Інструменти моделювання 26
2.3 Верифікація цифрових систем 27
2.3.1 Класифікація технологій верифікації 29
2.3.2 Технології моделювання 30
2.3.3 Статичні технології 32
2.3.4 Формальні технології 33
2.3.2 Підхід на рівні системи 34
2.4 Як впливає на розробку цифрових систем час виходу її на ринок 35
2.5 Висновок 37
3 ОГЛЯД МЕТОДОЛОГІЇ ПРОЕКТУВАННЯ СИСТЕМНОГО РІВНЯ 38
3.1 Вступ до методології електронних системних рівнів 38
3.2 Переваги ESL 39
3.3 Новітні технології дизайну 41
3.4 Мови системного рівня 42

3.5 Моделювання рівня транзакцій 43
3.5.1 Точність моделювання 44
3.5.2 Необмежений TLM 45
3.5.3 Проблема синхронізації 46
3.5.4 TLM на основі SystemC 47
3.6 Висновок 48
4 ФУНКЦІОНАЛЬНА ПЕРЕВІРКА З ВИКОРИСТАННЯМ АНАЛІЗУ ПОСИЛАНЬ 49
4.1 Проблема транзакційного аналізу даних 49
4.2 Транзакційний аналіз у Data Mining 51
4.3 Огляд аналізу посилань 52
4.4 Метод аналізу посилань для моделювання рівня транзакцій 54
4.5 Результати моделювання рівня транзакцій за допомогою FP дерева 57
4.6 Висновок 58
5 СЕРЕДОВИЩЕ МОДЕЛЮВАННЯ 60
5.1 Моделювання архітектури середовища 60
5.2 Рішення для перевірки «Riviera» 62
5.3 API моделювання 64
5.4 XML-інтерфейс для IP-адреса 67
5.5 Протокол передачі даних 69
5.6 Об'єктна модель ядра імітаційного моделювання 72
5.7 Основні характеристики системи 75
5.8 Висновок 75
6 JPEG-ДЕКОДЕР ПОТОКУ ДАНИХ 77
6.1 Опис проекту 77
6.2 Проблеми верифікації 78
6.3 Віртуальний прототип 80
6.4 Транзакційний аналіз даних 82
6.5 Висновок 88

ВИСНОВКИ 90
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 93
ДОДАТОК А 95
Графічний матеріал атестаційної роботи 95
ДОДАТОК Б 103
Зразок моделі транзактора 103

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ASIC – спеціальна інтегральна схема

ESL – електронний системний рівень

EDA – електронна автоматизація проектування

FPGA – польовий програмований масив воріт

HDL – мова опису обладнання

IEEE – інститут інженерів-електриків та електроніків

IP-Core – ядро інтелектуальної власності

HW/SW – апаратне / Програмне забезпечення

QoS – якість обслуговування

RTL – рівень передачі реєстру

SoC – система на чіпі

TLM – моделювання рівня транзакції

VHDL – дуже високошвидкісна мова опису обладнання для інтегральних мікросхем

XML – розширювана мова розмітки

ВСТУП

Реконфігуровані обчислення стосуються систем, що включають певну форму програмованості апаратного забезпечення - налаштування способу використання апаратного забезпечення за допомогою ряду фізичних точок управління [2]. Потім ці контрольні точки можна періодично міняти, щоб запускати різні програми, використовуючи одне і те ж обладнання. Перенастроюване обладнання пропонує гарний баланс між ефективністю та гнучкістю реалізації, як показано на малюнку 1-1. Це пояснюється тим, що реконфігурується апаратне забезпечення поєднує програмованість після виготовлення з просторовим (паралельним) стилем обчислень [2] специфічних інтегральних схем (ASIC), що є більш ефективним у порівнянні з тимчасовим (послідовним) стилем обчислення процесорів набору команд.

Завдяки зростаючим вимогам до гнучкості (наприклад, для адаптації до різних змінних стандартів та умов експлуатації), що пред'являються обчислювальними програмами, такими як бездротовий зв'язок, пристрої повинні бути високо адаптованими до запущених програм. З іншого боку, потрібна ефективна реалізація таких додатків, особливо в ресурсах, які вони використовують під час розгортання, де споживання енергії повинно обмінюватися на основі сприйманої якості програми. Суперечливі вимоги до гнучкості та ефективності реалізації не можуть бути задоволені звичайними процесорами набору команд та ASIC. Переконфігуроване обладнання у таких випадках представляє цікавий варіант реалізації.

Існують також інші причини, за якими використовувати переконфігурувані ресурси у розробці системи на мікросхемі (SoC). Зростаючі періодичні інженерні витрати (NRE) штовхають дизайнерів використовувати один і той же SoC у декількох додатках та продуктах для досягнення низьких витрат на чіп. Наявність реконфігурованих ресурсів дозволяє чітко налаштувати мікросхему для різних продуктів або варіантів

продуктів. Крім того, зростаюча складність у майбутніх конструкціях додає можливість включення проектних потоків, що може вимагати дорогого і повільного перепроектування мікросхеми. Реконфігуровані елементи часто є однорідними масивами, які можна попередньо перевірити, щоб мінімізувати можливість помилок у проектуванні. Крім того, здатність до програмуванняпісля виробництва дозволяє виправити або обійти проблеми пізніше, ніж із фіксованим обладнанням

1 ОГЛЯД ПРОБЛЕМИ ТА МЕТА РОБОТИ

1.1 Визначення проблеми

Сьогодні очевидно, що кожний проект, щоб досягти успіху, повинен бути добре перевірений. В результаті проект повинен відповідати вимогам, визначеним у функціональній специфікації. Крім того, помилки потрібно знайти та усунути на ранніх етапах проекту.

Проблема, яка виникає перед інженерами верифікації, полягає в тому, які стратегії та технології використовувати, щоб мінімізувати час перевірки та забезпечити необхідний стандарт якості.

Перевірка пристроїв System-on-Chip – це дуже трудомісткий процес. Це займає приблизно від 40 до 70 відсотків загального часу проекту. Тому ця проблема настільки важлива в дизайні мікросхем. На щастя, зараз у цій галузі доступні численні технології верифікації.

Зростання складності та кардинальне поліпшення в продуктивності дизайну ускладнюється і важче задовольнити вимоги якості проекту.

Ось чому в 1990-х дизайнери перейшли від роботи на рівні виходу до рівня передачі реєстру (RTL) [1]. Цей рух дозволив дизайнерам працювати на більш абстрактному рівні, який був більш зрозумілим, ніж логічний вихід.

Для цього були створені мови дизайну апаратних засобів (HDL). VHDL (Мова для опису апаратного забезпечення з дуже швидкою інтегральною схемою) та Verilog є найпопулярнішими серед них [2]. Ці мови визначені для роботи в RTL з точки зору машин кінцевого стану, арбітрів та елементів пам'яті. Мовна семантика дозволяє моделювати типові апаратні компоненти, такі як паралелізм, затримки, тактову частоту, порти та сигнали. Після того, як модель RTL описана в HDL, її можна моделювати з метою перевірки заздалегідь заданої функціональності та потім синтезувати до кремнію.

Однак, беручи до уваги збільшення розмірів дизайну та більше

функціональних можливостей, розміщених в одній мікросхемі, процес верифікації також повинен перейти на рівень абстракції від RTL до рівня транзакції. Ця мета спрямована на спеціальні мови моделювання на рівні системи (SLML). Однією із свіжих розробок у цій галузі є мова SystemC [3]. Вона створена на мові C ++ і має спеціальні конструкції, які дозволяють розробляти апаратні концепції для моделювання рівня транзакцій (TLM).

TLM працює на рівні функціональних запитів і пакетів даних, а не на рівні шинного циклу. Цей рівень допомагає дизайнеру чітким баченням поведінки системи. Це дозволяє проаналізувати швидке дослідження архітектури та аналіз апаратних/програмних засобів (HW/SW), спрямованих на виявлення оптимальної структури дизайну.

Сьогодні аналіз тенденцій ринку показує зростаючу необхідність інструментів системного рівня для проектування, верифікації та синтезу. За даними Mentor Graphics Survey [4], інженерам потрібна можливість опису, проектування та аналізу на високому рівні абстракції, спрощуючи процес створення цифрових систем.

Серед цільових архітектур очевидна тенденція, що незважаючи на сучасну популярність FPGA (Filed Programmable Gate Array), що в середньому складає близько 56% загального обладнання для впровадження System-on-Chip (SoC), використання постійно зростає - від 21% до 26% у майбутніх проектах. Це говорить про зростаючу складність проекту та набуває важливість підходу на рівні системи.

Основні механізми вдосконалення методології проектування:

- підвищення швидкості моделювання;
- співперевірка HW / SW;
- аналіз ефективності;
- проектування та верифікація на рівні системи.

Бажання респондентів відображають потреби створення віртуальних прототипів, які допоможуть прискорити процес моделювання, забезпечуючи попередню перевірку програмного забезпечення та архітектурний аналіз.

Найважливіші особливості методології системного рівня:

Моделювання системи, що складається з:

- моделювання повної системи;
- перевірка рівня транзакцій;
- аналіз ефективності.

Впровадження, яке включає:

- простий імпорт існуючих RTL IP;
- перевірка еквівалентності;
- синтез високого рівня.

Дивлячись на мовні переваги, можна зробити висновок, що найбільш важливим вибором для моделювання, перевірки та синтезу є мови на основі C. Що стосується апаратних мов, то тут спостерігається чітке зростання нових SystemC та SystemVerilog, а також класичного VHDL та об'єктно-орієнтованого C ++ у дизайні та синтезі. Понад 40% інженерів віддають перевагу синтезу з чистого C / C ++, а 25% - SystemC. У дизайні показник ще більше вражає: 50% для C/C ++ проти 64% для VHDL, лідера опитування.

Ось чому інженери верифікації хочуть працювати на системному рівні, де вони можуть працювати зі складними форматами даних, алгоритмами та протоколами. Їм потрібно думати і працювати на більш інтуїтивному рівні [5].

Тому ця робота необхідна задовольнити вимоги дизайнерів SoC, пропонуючи функціональну перевірку на системному рівні за допомогою сучасних мов високого рівня та методів верифікації.

1.2 Мета роботи

Метою даної роботи є розробка ефективного методу функціональної перевірки проектів SoC в рамках методології системного рівня. Це розслідування було розпочато під час мого стажування у верифікаційній компанії Aldec® і спочатку було пов'язано з TLM на основі верифікаційного

рішення Aldec® Riviera ™. Згодом область досліджень була розширена на системну перевірку проектів SoC.

В рамках цієї тези передбачається досягти наступних цілей:

- побудова моделі інтегрованого середовища перевірки на базі ядра TLM, яка забезпечить взаємозв’язок компонентів SoC;
- створення універсального XML-інтерфейсу для взаємозв’язку компонентів SoC;
- розробка алгоритму аналізу зв’язку результатів моделювання транзакцій;
- затвердження запропонованого рішення для верифікації демо-проекту.

Це лише короткий перелік необхідних дій для створення ефективного рішення для верифікації. Вважається, що цей список буде доповнено до кінця роботи.

1.3 Огляд літератури

Тема розробки та верифікації цифрових систем є дуже популярною в наш час. За останнє десятиліття проблема верифікації набула особливої актуальності із збільшенням розміру та складності конструкції. Ось чому провідні в галузі компанії беруть участь у виданні нових книг для з'ясування проблеми дизайну та верифікації мікросхем.

У роботах [1] та [2] розглянуто проблему класичного проектування та верифікації на RTL, можливості використання HDL для конструкцій ASIC та FPGA. Незважаючи на вже викладену інформацію, надаються базові принципи проектування та перевірки.

Детальний звіт про потреби сучасних інженерів представлений Mentor Graphics в європейському опитуванні [4]. Він відображає найважливіші нині проблеми дизайну, моделювання та синтезу складних проектів. Підняття на системний рівень розглядається як рішення.

Геніальна книга Пола Вілкокса [5], присвячена концепції професійної перевірки. Показані основні компоненти успішної перевірки. Запропонована ідея заснована на уніфікованій моделі верифікації Cadence, і вона є кращою лише для середовища розробки та верифікації Cadence.

У [6] рівні схеми розглядаються питання проектування компонентів SoC. Показано основні принципи використання статистичних моделей для моделювання пристроїв низького рівня. Проблеми моделювання на рівні системи досі не розкриті.

Переваги платформенно-орієнтованого підходу наведені в [7]. Незважаючи на гнучкість витоків, це дозволяє швидко розвивати цифрову систему, використовуючи при цьому високий рівень абстракції дизайну. Застосування мови розширюваної розмітки (XML) показано як невід'ємний компонент в рамках моделювання SoC.

Автори [9] описують різні методики верифікації, звертаючи увагу на перевірку рівня системи та проблеми міграції тестових стендів на різні рівні абстракції.

Огляд базової функціональної перевірки за допомогою TLM наведено в роботі [10]. Введено основні принципи побудови моделей TLM, а також методи транзакційного взаємозв'язку.

В роботі [13] пояснюється вплив ринку на потік дизайну мікросхем. Він містить економічну модель корпорації ATEQ із затримкою впровадження товару. Також підкреслюється важливість своєчасного випуску продукту. Підняття абстрактного рівня пропонується як рішення.

Бачення системного проектування компанією ARM представлено в роботі [15]. У статті описано всі переваги застосування нових методологій, таких як попередня розробка програмного забезпечення, архітектурний аналіз, налаштування та перевірка TLM, застосовані в інтегрованому середовищі проектування та верифікації ARM.

Роль SystemC у дизайні системного рівня показана в [8], [12], [14], [20], [21], [22] та [23]. Опис того, як SystemC може бути використаний для TLM,

наведено із введенням інструментів та короткою історією. Згадано питання вбудованої системи дизайну за допомогою TLM. Також вказується архітектурний аналіз та налагодження транзакцій. Відповідні стандарти SystemC від різних постачальників наведені в [15], [16] та [17].

У книгах [24] та [25] розповідається про здобуття даних та основи відкриття знань. Описані різні методи та підходи для обробки даних, включаючи аналіз зв’язків, який може бути застосований для майнінгу шаблонів у величезних масивах даних моделювання.

У роботах [26] та [27] пропонується реалізація методів видобутку частотних моделей. Наведено порівняльний аналіз різних алгоритмів видобутку шаблонів. Здатність вирішувати різні проблеми в різних областях за допомогою здобуття малюнків показана в цих книгах. Обробка даних моделювання не є винятком.

Опис новітнього стандарту System C 2.2 IEEE 1666, який підтримуєтсья Aldec, наведено в матеріалах [28] та [29]. Висвітлено всі нові функції продукту та переваги їх використання для TLM. Показана можливість використання симулятора Aldec для побудови середовища, що базується на TLM, проектування та верифікації.

Функціональні можливості та можливості реалізації процесу декодування JPEG пояснені в [32]. Пропонується процес декодування розділення процесу без підтримки TLM. Таким чином, ця модель може бути представлена як приклад підйому до системного рівня шляхом прийняття TLM.

На закінчення в цьому розділі представлені науково-технічні публікації провідних галузевих компаній у сфері автоматизації електронного дизайну, пов'язані з передовими технологіями проектування та верифікації. Розглянуті джерела не пропонують єдиний підхід у розробці SoC, але показують різноманітність потужних методів підвищення якості продукції за короткий проміжок часу.

1.4 Організація роботи

Розділ 1 – це вступ до цієї тези, містить короткий виклад сучасних технологій в розробці та верифікації мікросхем. Огляд літератури, мету роботи та організацію.

У главі 2 описані загальні питання розробки та верифікації мікросхем. Представлено сучасні тенденції та засоби моделювання. Запропоновано класифікацію різних технологій верифікації та пояснено вплив ринку на потік проектування.

Огляд нової методології системного рівня обговорюється в главі 3. Описані мови системного рівня, включаючи стандарт SystemC, як принципи моделювання рівня транзакцій.

Розділ 4 пояснює проблему моделювання аналізу результатів. Запропоновано новий алгоритм аналізу зв’язків величезних даних моделювання з метою інтеграції його в середовище моделювання.

Детальний опис середовища моделювання наведено в главі 5. Він охоплює архітектуру середовища, інтерфейс моделювання, протокол передачі даних, об'єктну модель та огляд основних функціональних можливостей системи. Висвітлено основні особливості системи. Серед них інтерфейс взаємозв'язку для включення блоків SoC та новий алгоритм моделювання аналізу результатів зв'язку.

У главі 6 представлені результати моделювання та верифікації проекту декодера JPEG з використанням створеного середовища моделювання.

У главі 7 висновки проведеної роботи та пропонуються можливі напрями майбутніх досліджень.

2 ПРОЕКТУВАННЯ ТА ПЕРЕВІРКА ЦИФРОВИХ СИСТЕМ

2.1 Вступ до проектування цифрових систем

Найважливішою частиною більшості сучасних електронних пристроїв є інтегральні схеми (IC). Цифрові системи намагаються реагувати на зростаючий попит на високу продуктивність і низьку вартість, вводячи все більше і більше функціональних можливостей у чіп. Є кілька переваг використання мікросхем порівняно з використанням дискретних компонентів, таких як транзистори в схемі:

- висока щільність штампу;
- надійність;
- низька вартість.

Дизайн мікросхем стає складним завданням з урахуванням того, що цифрові IC складаються з мільйонів транзисторів. Інструменти потрібні на багатьох рівнях процесу проектування для імітації схеми, перевірки обмежень, створення остаточної верстки тощо. Такі програмні засоби зазвичай називають інструментами електронного проектування (EDA) в [1] та [2].

Сучасна цифрова система включає як: апаратні, так і програмні компоненти. Ці компоненти повинні бути випущені для споживача при певному рівні продуктивності та якості. IC працюють як окремі або комбіновані частини електронної системи, наприклад, мікропроцесори, пам'ять, підсилювачі або генератори. IC поділяються на три категорії за призначенням:

- аналог;
- цифровий;
- змішаний.

Невеликий розмір реалізації кремнію дає значне поліпшення

продуктивності системи порівняно зі старою системою технології плати. Висока швидкість, низьке енергоспоживання, скорочення часу виходу на ринок та вартість виготовлення є ключовими перевагами IC.

У 1965 році Гордон Мур прогнозував, що "кількість транзисторів, включених до IC, збільшуватиметься вдвічі" [6]. У світлі зростаючої складності цифрових систем постає нова проблема розміщення великої ємності транзисторів. Для її вирішення задумана нова методологія включення декількох електронних компонентів у мікросхему. Нова парадигма називається System-on-Chip (SoC) і означає побудову всієї цифрової системи на мікросхемі, що включає набір апаратних та програмних частин.

SoC здатний інкапсулювати досить складну електронну систему на одному чіпі. Він заснований на загальноприйнятих блоках для багаторазового використання - ядрах інтелектуальної власності (IP) - розроблених для виконання завдання.

SoC складається з різних компонентів, таких як процесорний блок (MPU), функціональні блоки для виконання спеціалізованих алгоритмів, блоки пам'яті, периферійні блоки вводу / виводу та інші цифрові та аналогові компоненти. Спеціальний інтерфейс з'єднання, такий як спільна шина, складається з метою встановлення з'єднань між блоками IP (рисунок 2.1).

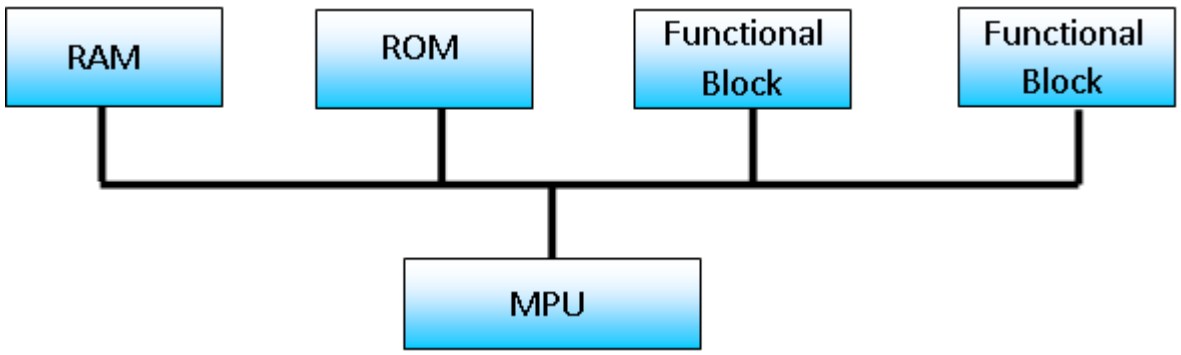

Рисунок 2.1 – Класична архітектура системи на мікросхемі

Щоб зробити процес розробки SoC швидшим і простішим, була створена так звана платформа SoC для створення шаблону SoC, специфічного для додатків. Такий підхід дає необхідну гнучкість та диференційованість з метою створення певної товарної лінійки. Таким чином, чіп може бути поставлений на ринок за короткий термін, скорочуючи час проектування та зусилля інженерів.

Найновіша додаткова технологія напівпровідника оксиду металів (CMOS) дозволяє платформі SoC складати вбудоване програмне забезпечення і навіть запускати його на багатоядерних процесорах в межах одного SoC.

Технологія «System-on-Chip» є значним проривом у напівпровідниковій промисловості. Це дозволяє реалізувати велику функціональність у невеликих розмірах кремнію, забезпечуючи високу продуктивність для вирішення складних проблем. Тому SoC можна використовувати в портативних пристроях, що реалізують всю систему на одній матриці.

Безперечно, майбутнє напівпровідникових галузей на базі SoC. Вважається, що аналіз потоку дизайну SoC може виділити поточні вузькі місця SoC для підвищення продуктивності та що можна зробити, щоб зменшити ці бар'єри.

2.2 Сучасні тенденції в дизайні цифрових систем

Існує два найпоширеніших типи чіпів. Перший тип - це специфічна інтегральна схема для програм (ASIC), коли функціональність мікросхем реалізована апаратно. Він має спеціальний додаток, і його не можна перепрограмувати на інші функціональні можливості, а також відмінити для копіювання його функціональності.

Другий - це програмовані польові масиви виходу (FPGA). Цей тип можна перепрограмувати. Ця технологія є досить гнучкою і може впоратися з досить складними проектами. Незважаючи на те, що вона може задовольнити

широкий спектр проектних вимог, ця методика є тижнем для складних систем на мікросхемі [2].

Для того, щоб чіп легко було налаштовано, вбудоване програмне забезпечення ввімкнено. Це дозволяє реалізувати більшу складність в мікросхемі. Функціонал, представлений у програмному забезпеченні, можна перепрограмувати простим способом. Програмний код може зберігатися в пам'яті та виконуватись процесором загального призначення або процесором сигналу.

SoC зазвичай набагато складніший ніж ASIC і є сучасним трендом у дизайні мікросхем. Типовий SoC інтегрує багато блоків, включаючи периферійні IP-адреси, шини, складні з'єднання, кілька процесорів, пам'ять, тактову частоту та розподіл потужності, тестові структури та шини відповідно до [6]. У шини завжди є кілька головних блоків, в результаті чого архітектури шин є складними. Складність нових SoC, що розробляються або плануються на найближче майбутнє, постійно зростає.

Ця нова методологія включає традиційні підходи до проектування, такі як вбудоване програмне забезпечення, аналогові та змішані компоненти. Таким чином, інструменти проектування та верифікації повинні працювати з декількома технологіями або інтегрувати існуючі рішення, щоб швидше та з належною якістю отримати SoC. Вони повинні справлятися зі збільшенням розмірів і складними структурами для реалізованих алгоритмів.

2.2.1 Спільне проектування апаратно-програмного забезпечення

Як було зазначено раніше, дизайн SoC виглядає дуже перспективно, проте існують певні проблеми. Ці виклики забезпечують широкі напрямки досліджень. Один з них - спільне проектування обладнання та програмне забезпечення.

Важливо підтримувати паралельний процес проектування обладнання та програмного забезпечення в межах проектування SoC. Певний рівень

сполучення повинен існувати для того, щоб знайти проблеми дизайну на перших етапах процесу.

Зіткнувшись з проблемою розділення в дизайні SoC, важливо вирішити її таким чином, щоб забезпечити інженерам найкращу роботу системи та перевірку на основі заданої специфікації проекту.

Мета - розділити компоненти системи на апаратне чи програмне забезпечення, щоб система, що отримується, забезпечувала оптимальну продуктивність. Як правило, хорошого розподілу можна досягти за рахунок функції витрат, яка враховуватиме обмеження специфікації, такі як час виконання, пам'ять даних, частота та інші властивості проекту.

Процес розподілу не має єдиного результату, і кожен розділ повинен бути проаналізований відповідно до критеріїв функціонування витрат. В даний час на ринку не існує повністю автоматизованих і досить хороших дизайнерських інструментів. Сьогодні ця проблема знаходиться в процесі активних досліджень.

Ще одне тісно пов'язане питання - це тестування компонентів, представлених на різних рівнях абстракції, і написаних за допомогою різних мов. Хоча ми можемо перевірити апаратне та програмне забезпечення самостійно, важливо, щоб тестування проводилося одночасно. Для цього дуже важливо мати добре перевірений апаратно-програмний інтерфейс.

Прихований ризик зміни специфікацій слід враховувати у випадку, коли апаратну частину вже відкладено. У цьому випадку це може різко уповільнити продуктивність чіпа.

2.2.2 Класичні напрямки дизайну

Проектний етап - це сувора інженерна методологія створення, перевірки, затвердження та постачання кінцевої конструкції до виробництва, при чітко контрольованому рівні якості відповідно до [7].

Традиційно цифрові вбудовані електронні системи використовують напрямки проектування, як показано на рисунку 2.2.

Рисунок 2.2 – Класичні напрямки дизайну

Дизайн починається із загальної специфікації замовника. Після розподілу він розбивається на два різних напрямки діяльності: розробка апаратного забезпечення системи, розробка програмного забезпечення системи. Між цими двома шляхами дизайнерської роботи взагалі немає зв'язку, а апаратне та програмне забезпечення можна об'єднати лише на етапі інтеграції процесу проектування. Розробка обладнання, яке використовувалося для початку, виконується шляхом створення апаратних

моделей за допомогою мови опису апаратних засобів. Ці моделі пройдуть функціональну перевірку в симуляції, щоб перевірити правильність їх поведінки. Згодом синтез проводиться для отримання логічного списку. Після того як список зв‘язків буде готовий, він перейде до нижчих етапів проектування до фізичної перевірки. Після цього апаратне проектування - це, по суті, кінцева версія, готова до створення прототипу виготовлення системи.

Відповідно до класичного дизайнерського потоку програмні та апаратні блоки розробляються самостійно. Немає ніякого зв’язку між апаратними та програмними командами щодо тестування функціональності всього проекту. Хоча кодування можна запустити раніше. таким чином тестування програмного забезпечення, що отримує доступ до нових периферійних IP-адресів, вимагає порівняння RTL-кодів на емуляторі. Це коштовний процес із залученням дорогого обладнання. Гірша ситуація чекає тестовий чіп зі стадії виготовлення, щоб протестувати його на прототипній платі. Як результат, програмне забезпечення завжди перевіряється пізніше апаратного забезпечення.

Увімкнення прототипування в потоці проектування можливо здійснити апаратне/програмне забезпеченням. Якщо процес перевірки не виконаний, процес буде повторюватися, як зазначено на рисунку 2.2, поки не буде досягнуто необхідної функціональності системи. Нарешті, дизайн направляється на виготовлення.

Для підвищення якості розробленого продукту необхідно підтримувати обидві галузі розвитку, забезпечуючи процедури перевірки. У цьому випадку ми можемо знайти помилки набагато раніше, перш ніж апаратна частина буде впроваджена в чіп.

Сьогодні галузь SoC потребує інструментів спільного моделювання, які зможуть імітувати загальний проект за короткий проміжок часу.

2.2.3 Інструменти моделювання

Зараз у сфері EDA поширеним способом опису системи на мікросхемі є HDL, змінивши на застаріле представлення схеми. Мови HDL, найбільш популярні є Verilog та VHDL, в основному застосовуються при розробці цифрових компонентів електронних пристроїв.

Дані мови мають семантику, яка дозволяє моделювати типові апаратні компоненти паралелізму, затримок, тактових частот, портів та сигналів. Як тільки модель описана на HDL, її можна моделювати для перевірки заздалегідь визначених функціональних можливостей, а потім для їх синтезу в апаратному поданні.

Цей підхід застосовано до розробки тільки пристроїв, які містять лише апаратні компоненти; однак це не підходить для проектів SoC. Є кілька проблем, які виникають:

Типові системи можуть бути представлені на високих рівнях абстракції. Наприклад, мови на базі C зазвичай використовуються для представлення високо абстрактних систем. Через високу швидкість моделювання функціональна перевірка переважно проводиться на високому рівні абстракції системи (перевірка моделей C/C ++ набагато швидша, ніж моделі, представлені на HDL);

Здебільшого симулятори HDL не підтримують апаратно-програмне спільне проектування та співперевірку;

Добре перевірені IP-адреси, написані на C/C ++, для більшості використовуваних систем доступні на ринку, і їх використання може різко скоротити час проектування чіпа.

Через обмеження мови HDL для проектування SoC надаються спеціальні мови для моделювання на системному рівні.

Останніми мовами, які запроваджуються та підтримуються EDA, є SystemVerilog та SystemC [8]. Остання зроблена на C ++ і має спеціальні конструкції, що дозволяють моделювати концепції, пов'язані з обладнанням.

Останнім часом такі мови як C ++ та Java використовувались в інтересах об'єктно-орієнтованого проектування. Ці концепції дозволяють дизайнеру визначити систему на верхньому рівні опису і, таким чином, підвищити продуктивність, читабельність та можливість повторного використання.

Однак усі ці мови страждають від нестачі понять для перевірки і, отже, не дозволяють усунути згадані вище проблеми.

2.3 Верифікація цифрових систем

Сучасні системи на мікросхемі представляють складні системи з рівнем інтеграції, розміром до десятків мільйонів виходів. Раніше кожен з цих компонентів був розроблений окремо за допомогою різних систем EDA, і взаємозв'язок відбувався лише на завершальній фазі проектування.

Для сучасних проектів величезного розміру такий підхід ризикований: час на стабілізацію системи після взаємозв'язку компонентів заздалегідь складно оцінити, окрім помилок ранньої фази проектування можна виявити занадто пізно. У зв'язку з цим існує потреба у верифікаційних рішеннях, які дозволяють одночасно перевірити всі компоненти SoC на ранніх етапах проекту за допомогою прототипування віртуальних пристроїв, укладених у [9].

Крім того, надійність та ефективність повинні бути забезпечені якомога раніше та на більш високому рівні проекту. Є думка, що це дуже важка і трудомістка робота. Навіть крихітна помилка може усунути проблему із збільшенням витрат на виправлення в міру розвитку дизайну.

Мета перевірки - забезпечити, щоб конструкція відповідала функціональним вимогам, визначеним у функціональній специфікації. Деякі проблеми, які виникають, - це те, наскільки достатньо перевірки, які стратегії та технологічні варіанти використовувати та як мінімізувати час перевірки.

Підвищити рівень абстрагування проекту необхідно з наступних причин:

- проектування на більш високому рівні абстракції дозволяє нам будувати легкі високоскладні проекти;
- переконатись, що прототип високого рівня все ще правильний після перетворення дизайну з нижнього рівня;
- з метою використання конструкцій неоднорідного характеру (наприклад, апаратне / програмне забезпечення спільного проектування, аналогові та цифрові компоненти);
- вимоги до більш високої надійності системи для функціонування рівня чіпа в системному середовищі.

Для усунення вузьких місць верифікації вважається, що найкращим рішенням буде підвищення продуктивності верифікації. І тут концепція абстракції виходить із області дизайну, де вона застосовувалася для вирішення тієї ж проблеми. Для перевірки мікросхем було запропоновано мови вищого рівня, такі як C/C ++, SystemC та SystemVerilog. Однак конструкції, які використовувались для перевірки, не синтезуються і, отже, не використовуються дизайнерами як частина фактичного коду дизайну.

Запис режиму на високому рівні абстракції дає можливість представляти поведінку апаратних компонентів для функціональної перевірки. З підвищенням рівня абстракції нова методологія дозволяє збільшувати швидкість моделювання, щоб скоротити час перевірки.

Ідеальною формою представлення апаратних моделей є мови високого рівня, такі як SystemC, C/C ++. Найкращий спосіб перетворити алгоритмічну поведінку на реалізацію обладнання - це засоби синтезу. Вони перекладають алгоритмічні моделі в апаратні екземпляри, використовуючи попередньо визначену виробником бібліотеку компонентів.

В даний час розробка інструментів синтезу є найгарячішою темою розробки та верифікації цифрових систем. Сучасні інструменти, доступні на

ринку, дозволяють представити моделі SoC на високому рівні лише шляхом дотримання спеціальних вимог стилю кодування поточного постачальника.

2.3.1 Класифікація технологій верифікації

Ці питання кидають виклик як інженерам верифікації, так і постачальникам рішень для верифікації. У галузі існує широкий спектр варіантів технологій верифікації [9]:

- імітаційні (динамічні) технології;
- статичні технології;
- формальні технології;
- фізична перевірка та аналіз.

Нині доступні параметри технології підтвердження описані нижче. Для досягнення цілей верифікації SOC необхідно використовувати комбінацію цих методів (рисунок 2.3).

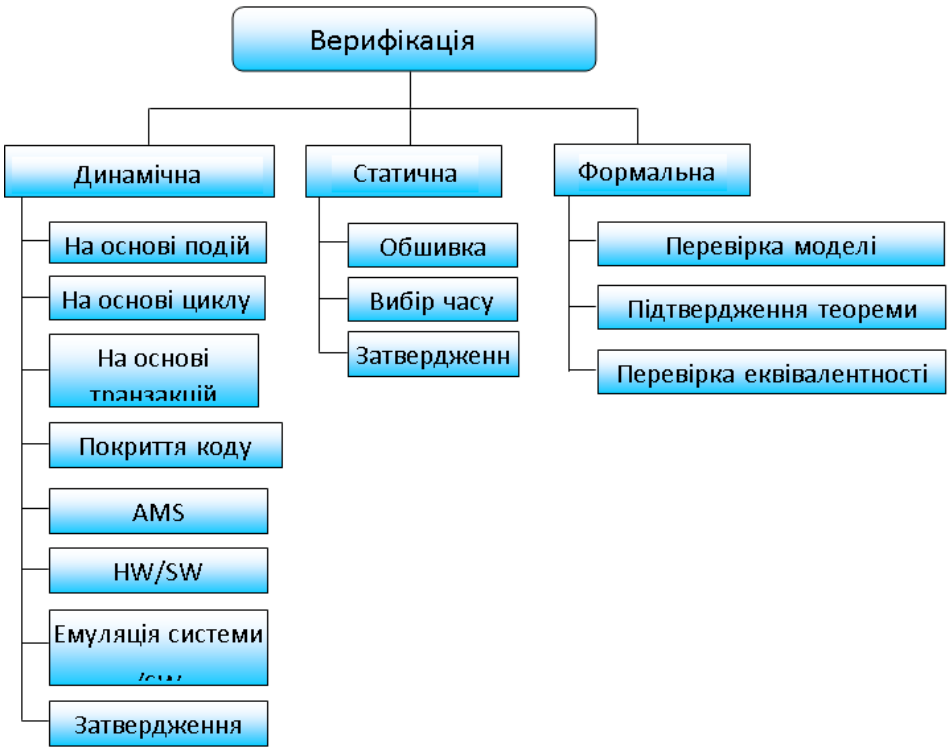

Рисунок 2.3 – Технології перевірки

2.3.2 Технології моделювання

Технології моделювання передбачають процес перевірки різних видів інструментів симуляції. Їм потрібно генерувати схеми введення та застосовувати їх протягом декількох тактових циклів до дизайну.

Симулятори на основі подій забезпечують точне середовище моделювання. Вони діють на основі подій, що охоплюють та доставляють їх до досягнення остаточного стану. Подія - це будь-яка зміна вхідного імпульсу. Таким чином, один компонент може бути оцінений кілька разів протягом одного циклу моделювання. Незважаючи на високоточне моделювання середовища, швидкість виконання залежить від розміру конструкції та рівня імітаційної активності. Це призводить до уповільнення складних конструкцій.

Циклічні симуляції забезпечують більш високу швидкість моделювання, ніж ті, що базуються на подіях. Вони не мають уявлення про час у тактовому циклі і оцінюють логіку між елементами стану та / або портами в одному кадрі. Оскільки кожен логічний елемент оцінюється лише один раз за цикл, це може значно збільшити швидкість виконання, але це може призвести до помилок моделювання. Симулятори на основі циклу функціонують лише в синхронній логіці.

Перевірка на основі транзакцій дозволяє моделювати та налагоджувати дизайн як набір одночасних процесів, які обчислюють та представляють поведінку модулів на рівні транзакцій [10]. Ці модулі обмінюються зв‘язком у вигляді транзакцій через абстрактний канал. Перевірка на основі транзакцій не вимагає детальних контрольних стендів з великими векторами. Цей тип перевірки різко збільшує продуктивність, дозволяючи здійснювати самоперевірку та спрямоване випадкове тестування.

Аналіз покриття кодом, який використовується для виконання на рівні RTL, забезпечує можливість кількісного визначення функціонального покриття, якого досягається певним тестовим набором, застосовуючи до

конкретної конструкції. Він оцінює різні рівні дизайну: індивідуальний блок або також повний чіп. Інструменти аналізу дають значення для процентного покриття кожного оцінюваного атрибуту та список неперевірених для частково перевірених областей проекту.

Співперевірка HW/SW базується на таких принципах, як спільне проектування HW/SW, про що вже говорилося вище. Вони одночасно виконують інтеграцію та перевірку апаратних та програмних засобів. Середовища спільної перевірки включають також емулятори програмного забезпечення або налагоджувачі, які використовуються командами з розробки програмного забезпечення. Це дає змогу команді програмного забезпечення виконувати програмне забезпечення безпосередньо на апаратній конструкції. Більше того, конструкція обладнання стимулюється реальним стимулом для введення, тим самим зменшуючи зусилля, необхідні для створення тестових стендів обладнання.

Емуляційні системи - це спеціально розроблені апаратні та програмні системи, які зазвичай містять конфігурувану логіку. Деякі системи емуляції, доступні в галузі, містять високошвидкісні процесори масиву. Ці системи запрограмовані на прийняття поведінки цільової конструкції і можуть імітувати її функціональність настільки, що вона може бути безпосередньо підключена до системного середовища, в якому кінцева конструкція призначена для роботи. Оскільки ці системи реалізовані в апаратному забезпеченні, вони можуть працювати на швидкостях, які на порядок швидше, ніж програмні симулятори, а в деяких випадках можуть наближатися до цільової швидкості проектування. Aldec, Inc. запропонувало одне з уніфікованих рішень для прискорення налагодження під назвою HES, що забезпечує максимальну ефективність моделювання та прискорює перевірку дизайну ASIC та FPGA на 10x-50x порівняно з традиційними методами [11].

Такий тип систем дозволяє також робити швидкі системи прототипування. Вони можуть бути використані для точного моделювання

прототипу передбачуваного SoC. Вони являють собою апаратні уявлення про дизайн, що перевіряється. Згідно з таким підходом компоненти SoC встановлюються на дочірні плати, які підключаються до системної материнської плати, що містить власні програмовані пристрої для з'єднання, що моделюють цільове з'єднання системи. Крім того, це може розглядатися як апаратний прискорювач, який дає можливість відображати деякі або всі компоненти програмного моделювання на апаратній платформі, спеціально розробленій для прискорення певних операцій моделювання.

Технологія аналогового змішаного сигналу (AMS) забезпечує опис та моделювання цифрових, аналогових та змішаних сигнальних систем в одному проекті. Він використовує спеціальні розширення мови, такі як Verilog-AMS та VHDL-AMS. Однак через складний характер аналогових компонентів їх перевіряли самостійно. Перевіряється лише інтерфейс після загальної інтеграції SoC.

2.3.3 Статичні технології

Перевірка обшивки проводить статичну перевірку поширених помилок дизайнера на етапі введення дизайну. Цей метод заснований на програмованих правилах і може використовуватися в парі з будь-яким сучасним компілятором HDL. Типи помилок, на які поширюється ця технологія, - це невідповідність між моделями рівнів RTL та рівня затворів, складне кодування для синтезу, погана розробка дизайну для тестування та навіть стилі кодування. Перевірка може проводитися на початку циклу проектування. Він ідентифікує прості помилки в дизайнерському коді, які забирають багато часу, щоб виявити їх за допомогою більш досконалих інструментів. Для перевірки всіх вимог до термінів надається статична перевірка часу. Він визначає, чи виконуються вимоги до термінів. Особливо, коли конструкція досить складна і кожен вхід може мати кілька джерел. Тому час може змінюватись в залежності від стану роботи схеми.

2.3.4 Формальні технології

Часто буває дуже важко виявити помилки, які залежать від конкретних послідовностей подій. Ці помилки можуть мати серйозний вплив на проектний потік, коли вони не виявлені на початку фази перевірки. Офіційні методи верифікації не вимагають тестових стендів або векторів для перевірки. Вони теоретично обіцяють дуже швидкий час перевірки та 100% охоплення.

Офіційними методами перевірки є:

а) Офіційна перевірка моделі:

- затвердження;
- властивості;

б) техніка доведення теореми;

в) формальна перевірка еквівалентності.

Офіційна перевірка моделі використовує формальні математичні прийоми для перевірки поведінкових властивостей конструкції. Інструмент перевірки моделі порівнює поведінку проектування з набором логічних властивостей, визначених інженером. Властивості, які використовувались безпосередньо з проектних специфікацій. Офіційна перевірка моделі добре підходить для складних управлінських структур, таких як шини-арбітри, декодери, мости між процесором та периферією тощо.

Поточні інструменти, представлені на ринку, спрямовані на офіційну перевірку двома способами:

а) використання тверджень - обмеження дизайну, які конкретизуються та формулюються всередині самої конструкції за допомогою спеціально розроблених мов проектування / перевірки, таких як SystemVerilog, Open Vera;

б) використання властивостей - дозволяють конкретизувати властивості за допомогою мови властивостей, таких як PSL, Sugar.

Методика доведення теореми заснована на побудові доказу проектної поведінки за допомогою теорем. Ця методика показує, що конструкція відповідає функціональним вимогам. Цей метод ще вивчається.

Офіційна перевірка еквівалентності - це метод доведення еквівалентності двох різних поглядів одного і того ж логічного проектування. Він заснований на математичних прийомах для перевірки еквівалентності еталонного проекту та модифікованого дизайну. Ці інструменти можна використовувати для перевірки еквівалентності цільової конструкції та еталонної конструкції, яка може представляти ідеальну модель, щоб довести правильність функціональності модифікованого дизайну. Але аналіз часу є все ж необхідним.

2.3.2 Підхід на рівні системи

Як уже згадувалося, дуже важливо розпочати процес перевірки якомога раніше. Таким чином, планування верифікації дизайну повинно починатись одночасно зі створенням специфікацій для системи відповідно до класичного потоку проектування. Це означає, що методологія повинна забезпечувати розробку та верифікацію зверху вниз. Вихідною точкою для будь-якої конструкції зверху вниз є функціональна специфікація, і поведінка системи моделюється за тим же принципом. Це може бути виконана специфікація, але зазвичай це письмова специфікація з усіма пов'язаними двозначностями природних мов. З функціональної специфікації розробляється детальний план верифікації.

Для перевірки правильності функціональності конструкції створюється спеціальний тестовий зразок поведінки. Найпоширенішими мовами є C / C ++, SystemC та SystemVerilog. Після перевірки поведінки системи система адаптується до відповідної архітектури, використовуючи апаратні та програмні IP-адреси, доступні в бібліотеці постачальника або записані як частина процесу проектування. Випробувальний стенд на рівні системи, як

правило, не враховує детально визначені цикли деталі системи, призначеної для проектування.

Перевірка на основі транзакцій може використовуватися для на рівні системи з метою перевірки додаткової функціональності загальної системи. Це включає перехід від моделі рівня транзакцій до моделі точного сигналу та моделі зв'язку протоколу.

2.4 Як впливає на розробку цифрових систем час виходу її на ринок

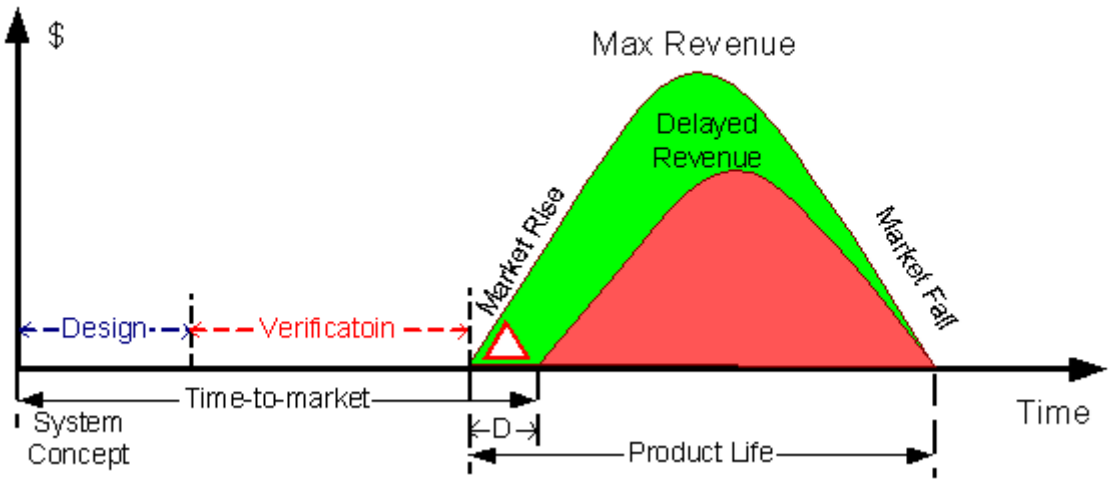

Рисунок 2.4 – Цикл життя проекту

У наш час інженери стикаються з надзвичайно складними проектами під тиском зменшення часу на ринок і більш жорстких обмежень витрат. [12] зауважив, що «час на ринок - це кількість часу, необхідного для перетворення ідеї в реальний продукт для продажу». Для отримання більшої частки ринку та отримання більшого доходу час на ринок потрібно скоротити. Тож продукт буде доступний раніше і потрапить на ринок.

Малюнок 2.4 ілюструє модель, що базується на баченні корпорації ATEQ [13], яка відображає економічний вплив затримки впровадження товару. Ця модель показує нам суму втраченого доходу (незатінений регіон у трикутнику максимального доходу) від затримки виходу на ринок. Він

враховує загальний прогнозований дохід продукту та тривалість ринкового вікна. Коли життєвий цикл товару короткий, запізнення на ринок часто означає високий ризик фінансових втрат, що призводить до кризи.

Зменшується не тільки час виходу на ринок для всіх областей застосування, але це супроводжується міграцією від традиційних військових та промислових застосувань до споживчих товарів, які мають найкоротший час циклу. У цьому випадку середній час циклу повинен скорочуватися швидше, ніж будь-який сегмент ринку. В іншому випадку ви можете пропустити ринкове вікно, наприклад, коли товар повинен бути доставлений певною подією.

Ці технологічні та ринкові проблеми мають драматичний вплив на методології та інструменти перевірки. За оцінками, від 40 до 70 відсотків загальних зусиль на розробку витрачаються на завдання верифікації. І приблизно 80 відсотків всього написаного коду знаходиться в середовищі перевірки. Очевидно, що ці заходи з верифікації повинні бути виконані ефективніше, щоб вирішити загальні ринкові виклики.

Для великих корпорацій слід враховувати кілька критеріїв, коли для вибору нового підходу до верифікації потрібно вибрати правильний інструмент перевірки. Серед них:

а) вплив нових технологій верифікації на вартість та час виходу на ринок;

б) інфраструктура CAD;

в) проектні потужності;

г) рентабельність інвестицій;

ґ) підтримка декількох географічних місць дизайну;

д) навички проектувальників та інженерів з верифікації;

е) використання системи.

2.5 Висновок

Вважається, що єдиним способом впоратися з небезпечними складними конструкціями реального світу є надання нової методології проектування та перевірки в потоці проектування за допомогою моделей високого рівня, написаних сучасними мовами опису дизайну. Було представлено деякі компроміси, які компанії могли зробити для того, щоб застосувати нові інструменти та технології для досягнення успіху на ринку в довгостроковому масштабі. Незважаючи на докази численних переваг переглянутого підходу, все-таки не так просто прийняти рішення про певний інструмент, мову чи технологію, який би відповідав вимогам верифікації та дизайну кожної компанії.

Бездротова мережа датчиків - це спеціальна мережа, що складається з великої кількості невеликих недорогих пристроїв, позначених як вузли (моти). Ці вузли - пристрої, що працюють на батареях, здатні спілкуватися між собою, не покладаючись на будь-яку фіксовану інфраструктуру.

3 ОГЛЯД МЕТОДОЛОГІЇ ПРОЕКТУВАННЯ СИСТЕМНОГО РІВНЯ

3.1 Вступ до методології електронних системних рівнів

Дизайн електронного рівня (ESL) - одна з найгарячіших тем у складному дизайні цифрових систем сьогодні. Незважаючи на ідею описати систему на високому абстрактному рівні, яка існує вже впродовж останніх років, лише зараз ця методологія знайшла своє впровадження в проектний потік. Системний рівень означає використання відповідного рівня абстракції для кращого розуміння системи з метою скорочення часу та витрат на розробку [14].

ESL надає інструменти та методології, які дозволяють дизайнерам описувати та аналізувати мікросхеми на високому рівні абстракції та мінімізувати таким чином загальні витрати на проектування електронних систем. ESL дозволяє описувати дизайн SoC досить абстрактно та швидко, щоб вивчити проектний простір та надати віртуальні прототипи для апаратного та програмного забезпечення. Цей підхід також підкреслює проблеми управління складністю, такі як повторне використання IP-адресу та причини перевищення рівня дизайну [15].

ESL-дизайн працює трьома способами:

- обмеження, отримані за специфікацією замовника та реалізовані на високому рівні абстракції, можуть поширюватися на нижчі рівні проектування в межах різних інструментів реалізації;
- процес верифікації проектування починається раніше, що допомагає краще оцінити інвестиційні вкладення;
- забезпечення розробки та інтеграції вбудованого програмного забезпечення.

3.2 Переваги ESL

Зростання ESL базується на виникаючих вимогах щодо скорочення критерію часу на ринок. За даними [4], 43% респондентів називають це основною причиною застосування методів ESL. Також інженери хочуть використовувати уніфіковані інструменти дизайну, мови на основі C на високому рівні для моделювання, перевірки та навіть синтезу. Крім того, сучасні проекти SoC складаються лише з 25% оригінального коду, ще 75% - це існуючі IP-блоки. Використовуючи методологію системного рівня, можна побудувати та моделювати цілі системи більш швидким способом, який пропонує класичний RTL. Залучення мов на основі C до процесу зв'язку специфікацій із апаратною реалізацією значно зменшує кількість помилок. Більше того, моделі ESL з високою абстракцією дозволяють дизайнерам систем будувати віртуальні прототипи для повного підтвердження своїх систем, що також дає більш безпечні конструкції.

Розробка віртуальних прототипів отримує раннє функціональне представлення SoC. Таким чином, вбудоване програмне забезпечення може записуватися і перевірятися декількома командами паралельно і перевірятися на віртуальному прототипі. Як результат, це скорочує час розробки та полегшує процес спільної перевірки апаратного та програмного забезпечення.

Інструменти ESL використовуються спочатку для моделювання, верифікації та валідації. Але сьогодні вони поширюються на інші частини дизайнерського потоку, де автоматичне створення апаратного та програмного забезпечення використовується для прискорення процесу проектування. Як було продемонстровано, потік даних від описів на основі C на високому рівні до RTL, а потім до рівня виходів забезпечується за допомогою блокової збірки в рамках методики системного рівня прийняття.

Зростаючий інтерес до впровадження системного рівня викликає вимогу до методологій проектування ESL, таких як моделювання системи з

моделюванням повноцінної системи, перевірка рівня транзакцій та аналіз продуктивності, впровадження системи з імпортом існуючих IP, перевірка еквівалентності, синтез високого рівня для деяких функціональних блоків. Архітектурне дослідження та аналіз потужності задіяні як у моделюванні, так і в впровадженні системи.

Ще один важливий момент – підхід орієнтований на платформу, який вимагає налаштування системи. Це означає можливість імітувати різні параметри, такі як різні ядра процесора, різні ширини шини, різні розміри пам'яті та конфігурації, різні програмовані двигуни та безліч сумішей периферійних пристроїв. За допомогою такої реалізованої платформи пристроїв можна за короткий час задовольнити зростаючі потреби клієнтів.

Як було сказано раніше, дослідження в ранньому дизайні є однією з особливостей методології ESL. Архітектурний рівень ESL - це відображення функцій на обчислювальні машини. Це перший етап розділення системної поведінки, і перевірка оптимальності цих варіантів є завданням, яке більшість людей вважає ESL.

Ця методологія дозволяє вивчити наслідки відображення її в архітектурі дизайну. Це дає можливість визначати, які функціональні блоки слід використовувати та чи потрібно будувати нові апаратні блоки IP, одночасно оптимізуючи схеми зв'язку, планування, нарізки та арбітражу.

Намагаючись моделювати ці блоки якомога раніше, необхідно забезпечити основні блоки IP, такі як процесорні ядра та шини, моделями високого рівня (написані мовами на основі C, такими як C / C ++, SystemC). Починаючи з моделей високого рівня, можна спочатку моделювати та реалізовувати нові конструкції. Ці моделі надають можливість моделювати функції системи в тисячі або мільйони разів швидше, ніж на рівні RTL, але компроміс - це точність синхронізації. Завдання полягає у пошуку балансу між швидкістю моделювання та достатньою точністю часу.

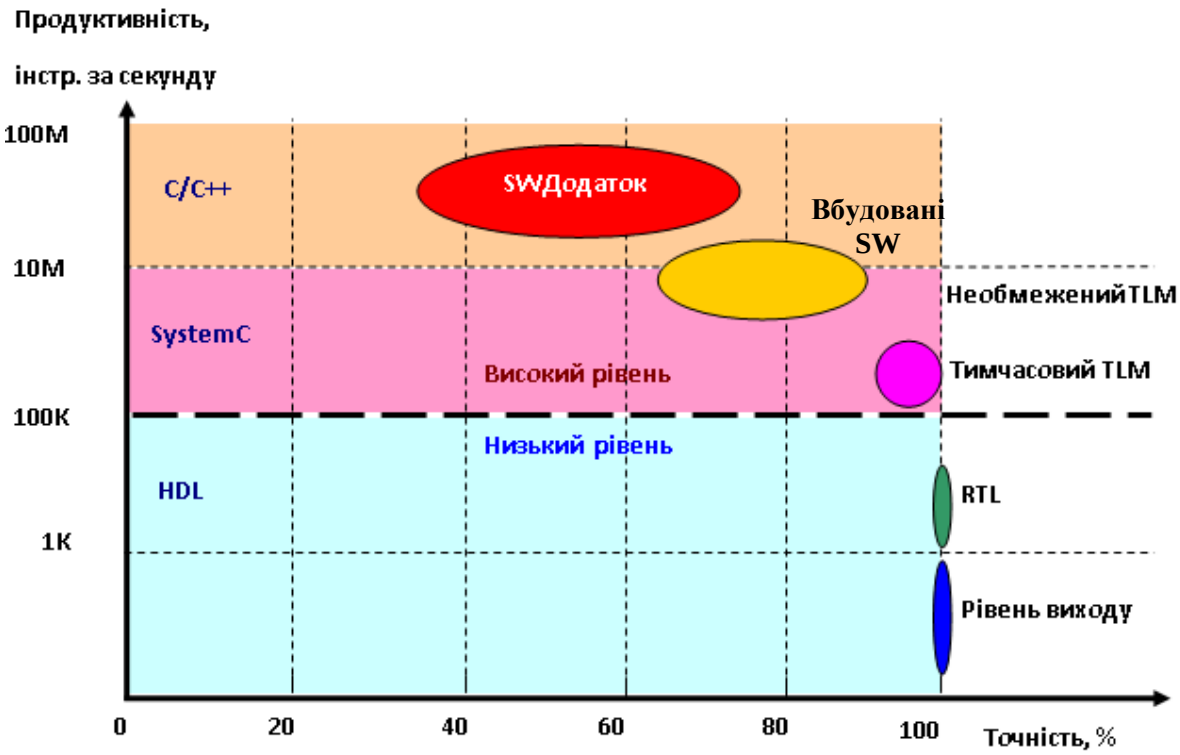

Рисунок 3.1 – Продуктивність (Точність компромісів)

3.3 Новітні технології дизайну

Отже, беручи до уваги нову методологію системного рівня, нові зміни надходять у проектний потік (рисунок 3.2). Дизайн все ще починається із специфікації замовника. Але після цього з'являється новий етап віртуального прототипування. І апаратне/програмне забезпечення розробляється одночасно на основі методології ESL.

Це дозволяє протестувати зібрану систему під час проектування. Це одна з найзнаменніших відмінностей між новим та класичним потоком дизайну SoC.

Після того, як віртуальний прототип буде доступний, можна було перевірити багато критеріїв, поки розпочато апаратне впровадження. У той час, коли перший тест-чіп готовий, досягається високий рівень впевненості, проводячи всебічну спільну перевірку. Новий дизайн SoC безумовно підвищить якість сконструйованого кремнію.

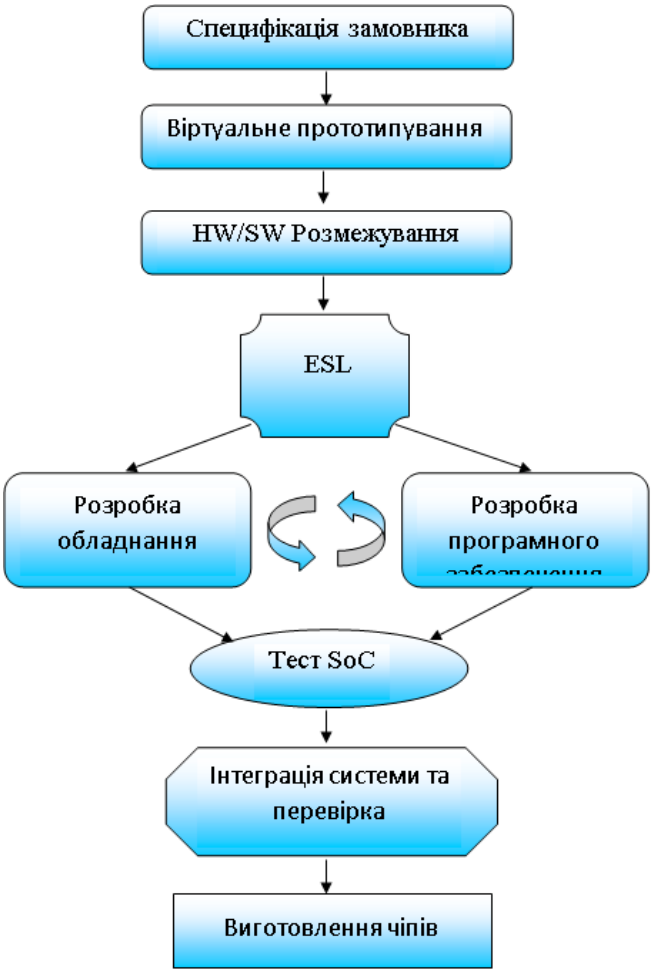

Рисунок 3.2 – Потік дизайну

3.4 Мови системного рівня

Зіткнувшись із проблемою побудови складних систем, що містять величезну кількість компонентів, включаючи вбудоване програмне забезпечення, існує потреба в мові моделювання, яка буде справлятися з розмірами та складністю цих систем.

Незважаючи на те, що найпоширенішим способом опису цифрових систем є HDL, процес моделювання рівня транзакцій вимагає мову високого рівня. Серед них: C \ C ++, SystemC [16], SpecC [17], Cynth з SystemC [18], VCC тощо. І найпоширенішою і широкою є SystemC. Це стандарт для транзакційного моделювання, який підтримується багатьма постачальниками EDA. Мова надає механізм управління складністю SoC із можливістю

спільного моделювання апаратного та програмного забезпечення на різних рівнях абстракції. Ця можливість недоступна в традиційних HDL.

SystemC - це відкрита бібліотека класів C ++, яка використовується для розробки та перевірки апаратних систем. Бібліотеки класів SystemC додають апаратні атрибути до мови C ++. Однією з головних переваг SystemC є те, що вона може бути використана для опису системи на кількох рівнях абстракції. Від дуже високого рівня функціонального опису до низького рівня синтезованого коду RTL.

Проект SystemC був розпочатий в 1999 році на вимогу кількох компаній EDA, серед яких CoWare, STMicroelectronics та Synopsys. Того ж року було оголошено Open SystemC Initiative (OSCI) для підтримки промислового стандарту для SystemC. Перша версія SystemC (SystemC 1.0) не мала конструкцій високого рівня, і в основному була орієнтована на RTL-дизайн. А в липні 2001 року випуск SystemC 2.0 включає нові функції високого рівня (нові канали, події та розумна швидкість моделювання моделей рівня транзакцій), які спонукають до зростання дизайну на системному рівні. Нові моделі основних компонентів SoC з’явилися за допомогою нового мовного стандарту, який набагато швидший, ніж еквівалентний цикл точних моделей RTL [19]. Серед них: модель шини, включаючи управління адресами, майстер шини, що створює передачі для читання / запису, пам'ять, таймер та контролер переривання.

3.5 Моделювання рівня транзакцій

У світлі ESL та відповідних мовних можливостей просто час розглянути по суті методику, яка допомагає реалізувати згадані вище переваги моделювання високого рівня [20]. Мабуть, що необхідно відокремити комунікацію від обчислювального процесу в модельованій системі.

Отже, потрібен підхід моделювання транзакцій на основі мов програмування високого рівня, так зване моделювання рівня транзакцій (TLM).

TLM означає моделювання модулів SoC, що представляє поведінку компонентів SoC. Зв'язок між модулями реалізується за допомогою спеціальних абстрактних каналів транзакцій. Модуль з'єднується з іншим за допомогою інтерфейсу зв'язку через порти модулів. У TLM взаємозв‘язок та обчислення відокремлені один від одного.

Транзакція - це подія передачі або синхронізації даних між двома модулями високого рівня відповідно до специфікації апаратних / програмних систем. Структура транзакції не має значення і може бути визначена як певна кількість бітів або повне зображення регістра або буфера пристрою, або навіть надання формату протоколу шини, що буде дуже корисно для арбітражних процесів в рамках дослідження архітектури SoC.

TLM допомагає впоратися з вузьким місцем SoC, забезпечуючи надійну методологію підтримки ранньої розробки програмного забезпечення та обладнання, архітектурного аналізу та функціональної перевірки. У майбутньому ця методологія призведе до численних переваг в продуктивності праці. Отже, вважається, що незабаром час виходу на ринок скоротиться. Як результат, TLM дає посилання на об'єднання зусиль з моделювання різних команд для досягнення синергії дизайну. Крім того, непогана ідея для міжгрупового спілкування, яке буде заохочуватися одночасною роботою з TLM. Нарешті можна було отримати ефективне управління проектами SoC.

3.5.1 Точність моделювання

Точність моделювання залежить від сучасного підходу моделювання та відображає точність моделі у поданні наміченої поведінки та діяльності даної системи. Ступінь точності моделювання визначається на основі двох

значущих критеріїв:

а) Точність даних - показує рівень шорсткості для даних і може змінюватись від пакету додатків до розміру шини;

б) Точність синхронізації - визначає часову поведінку моделі під час моделювання.

Як і будь-яка стратегія моделювання в потоці проектування SoC, для визначення точності моделювання потрібен підхід TLM. Відповідно до вищезазначених критеріїв, передбачаються два основні класи TLM:

- тимчасовий TLM;
- невчасний TLM.

Тимчасовий TLM містить більше часових анотацій для поведінкових та комунікаційних специфікацій відносно менш абстрактної моделі. Тому тимчасова точність TLM вимагає розробки програмного забезпечення та аналізу архітектури в реальному часі. Він розташований нижче в потоці проектування SoC і також відомий як перегляд програміста плюс терміни (PVT).

Необмежений TLM пропонує спосіб створення архітектурної моделі та забезпечує ранню функціональну перевірку без необхідних параметрів часу. Висока швидкість моделювання є ключовим рушієм цієї моделі. Оскільки необмежений TLM служить в основному програмістам, його також називають баченням програміста (PV).

3.5.2 Необмежений TLM

Як було зазначено вище, необмежений TLM задовольняє програмістів та інженерів верифікації на початку функціонального проектування та верифікації. Анотації щодо часу не є важливими на цьому невчасно визначеному рівні. З цієї причини інформація про взаємозв'язок та арбітраж не буде потрапляти в тимчасовий TLM.

Те, як внутрішні дані моделі, що розробляється, стають доступними - це інтерфейс програмування (API), який забезпечує спеціальне управління передачею даних.

Ще одним важливим питанням будь-якого підходу до моделювання є обчислювальна модель. У цьому контексті необмежений TLM не має жодної інформації про синхронізацію, наприклад, тактові сигнали. Отже, всі процеси виконуються одночасно для доступу до будь-якого з системних ресурсів. Тут існує проблема виконання паралельних процесів.

Для того, щоб задовольнити цю вимогу, необмежений TLM пропонує такі принципи: незалежне виконання процесів, паралельне виконання, синхронізація процесів, технологічний зв‘язок.

3.5.3 Проблема синхронізації

Вирішуючи задачу синхронізації для різних процесів у системі, слід визначати причинно-наслідкові зв’язки між ними, щоб дотримуватися залежностей. Тільки явна синхронізація системи може визначати порядок виконання таких процесів у всій системі.

Механізм синхронізації означає інформування інших частин системи про деякі зміни стану системи. У реальних апаратних схемах системні синхронізації моделюються за допомогою сигналів переривання, опитування або поштової скриньки.

Існує два типи синхронізації:

- «відсилання» - процес надсилає подію, яка може впливати на поведінку або стан інших процесів;

- «отримання» - процес очікує на вхідні події з системи.

Головною метою синхронізації є забезпечення послідовності даних, що захищають доступ до вмісту даних шляхом паралельного процесу у невизначеному стані.

Кожен з цих процесів дотримується певної послідовності (рисунок 3.3):

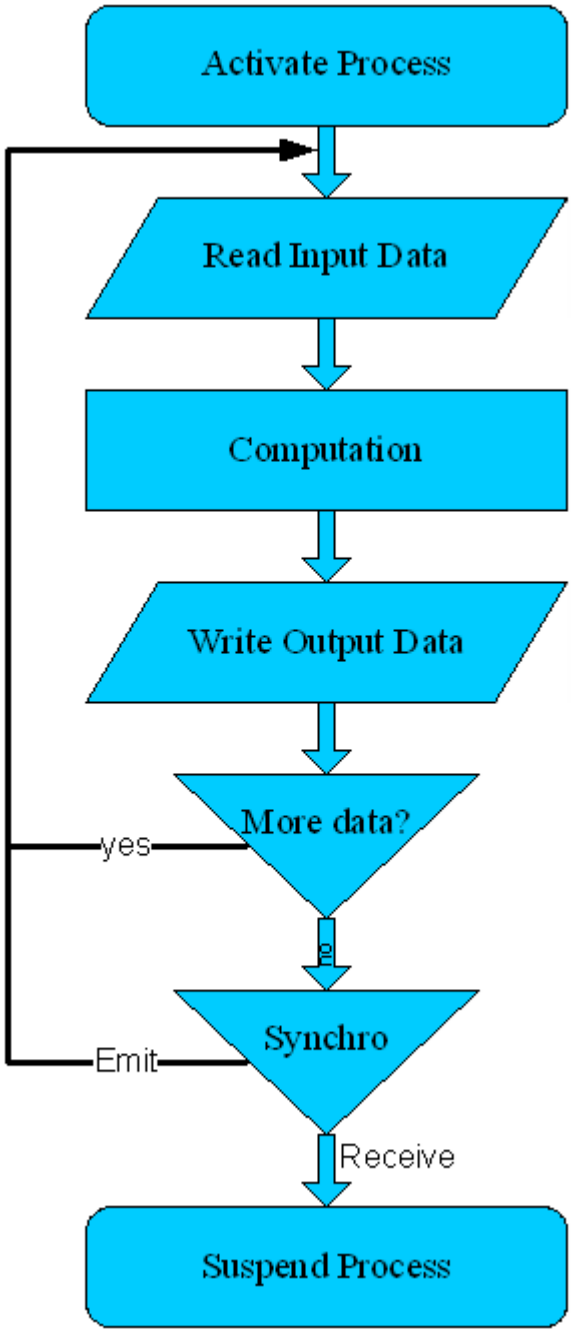

Рисунок 3.3 - Модель синхронізації

3.5.4 TLM на основі SystemC

Отже, SystemC 2.0 - ідеальний вибір для моделювання апаратних та програмних засобів. Оскільки розширення системного рівня в SystemC 2.0 підтримують TLM, вдосконалення зв'язку, моделювання виконуваних специфікацій, спільне проектування HW/SW. Будучи стандартною мовою для

проектування системного рівня, SystemC включає узагальнене моделювання зв'язку та синхронізації з каналами, інтерфейсами та подіями [21].

SystemC забезпечує об'єктно-орієнтовану методологію, яка базується на наборі класів C ++. Ідея мови полягає у запуску апаратних моделей разом з висококласними в одному ядрі імітації. SystemC також має можливість уточнювати низькорівневі, чітко визначені цикли та моделі RTL та інтегрувати існуючі високоякісні моделі C / C ++ [22].

SystemC включає підмножину можливостей мови C ++. Ось чому програмісти C ++ можуть працювати з SystemC комфортно, особливо коли джерела C ++ вже доступні. Використовуючи підхід на основі C, можна використовувати ті самі інструменти налагодження [23].

3.6 Висновок

На закінчення, методологія ESL з її численними перевагами забезпечує швидкий і простий спосіб проектування та перевірки моделей на системному рівні представлення. Ця методологія увімкнута за допомогою застосування технології TLM на основі сучасної мови дизайну SystemC.

Основними перевагами використання підходу на системному рівні є віртуальна прототипізація, висока швидкість моделювання з необмеженим TLM, транзакційна перевірка, імпорт HDL-моделей низького рівня в дизайн ESL, використовуючи мову на основі C високого рівня.

В результаті було запропоновано новий напрямок дизайну з віртуальними прототипами та можливостями спільного проектування обладнання та програмного забезпечення. Таким чином, розробка дизайну та виявлення помилок можливі на ранніх стадіях проектування.

4 ФУНКЦІОНАЛЬНА ПЕРЕВІРКА З ВИКОРИСТАННЯМ АНАЛІЗУ ПОСИЛАНЬ

4.1 Проблема транзакційного аналізу даних

Намагаючись включити методологію ESL за допомогою верифікації на основі транзакцій, щоб підвищити рівень абстракції верифікації, інженери неминуче зіткнуться з новою проблемою транзакційного аналізу даних.

Сучасні покоління SoC мають структуру безлічі головних / другорядних. Ось чому необхідна потужна система маршрутизації для з'єднання всіх цих блоків IP.

Однак є кілька недоліків. По-перше, складність існуючих систем маршрутизації, які продовжують швидко рости. Таке зростання унеможливлює проведення звичайного ручного аналізу трафіку та вивчення архітектури. Незважаючи на те, що аналіз вручну все ще корисний для вивчення основних випадків, для повного аналізу трафіку та вивчення архітектури необхідно використовувати потужний інструмент моделювання.

По-друге, система маршрутизації включає в себе величезну кількість IP-блоків змішаних протоколів під час інтеграції системи. Якщо проблема виникає в цей момент, потрібно буде перевірити всі компоненти платформи SoC одночасно. Це буде велика робота для інженерів, беручи до уваги кількість сигналів, які мають перевіряти форму хвилі. Незважаючи на кількість значущих, це менше 1% від загальної кількості сигналів. Наприклад, проста модель ESL декодера JPEG виробляє близько 57000 транзакцій за схемою зображення 2 Мрх. Потрібна велика робота, щоб проаналізувати кожен з них руками.

Для вирішення всіх цих проблем необхідно надати нову методологію транзакційного аналізу. Одне з існуючих рішень - транзакційне налагодження, основна ідея якого полягає в перетворенні сигнальних

комбінацій в єдину транзакцію з відповідною локалізацією інформації.

Методика налагодження транзакцій дозволяє зменшити зусилля налагодження в десятки разів. Більше того, такий підхід допомагає уникнути зайвого вивчення різних протоколів зв’язку та рівнів абстракції в системі. Це робить результати аналізу набагато зрозумілішими, ніж результати аналізу сигналів.

Дедалі більше зростаючий рівень абстракції, з'являється сувора ієрархія даних. Наступним рівнем представлення даних є потік транзакцій - набір транзакцій, що відбуваються в конкретному контексті. Наприклад, транзакції між двома маршрутизаційними з'єднаннями групуються як певний потік транзакцій. У нашому прикладі декодера JPEG з 57000 транзакціями є лише 3 значущих типи транзакційних потоків. Кожен потік складається приблизно з 5 транзакцій, таким чином, це дозволяє зменшити кількість записів у базі даних у п’ять разів. Розділивши транзакції на транзакційні потоки, можна зменшувати його кількість і, як результат, вирішити проблему верифікації простіше і швидше.

Операції можуть перекриватися в тому випадку, коли одна транзакція розпочинає свою передачу в потоці до того, як інші транзакції, попередньо збережені в тому ж потоці, закінчують свої передачі. Тому вказівки ієрархії між різними транзакціями можна визначити для відображення їх взаємозв'язків, таких як відносини попередник-наступник або батько-дитина (рис. 4.1). За бажанням можна перейти на рівень сигналів всередині кожної транзакції, тому ця інформація є узгодженою.

У результаті транзакційне налагодження підвищує рівень спостереження і таким чином спрощує взаємозв'язок або представлення зв'язку.

Для застосування налагодження транзакцій в аналізі SoC необхідні спеціальні алгоритмічні методи для обробки та зберігання великого обсягу даних. Навколишнє середовище повинно підтримувати запис, візуалізацію та аналіз транзакцій.

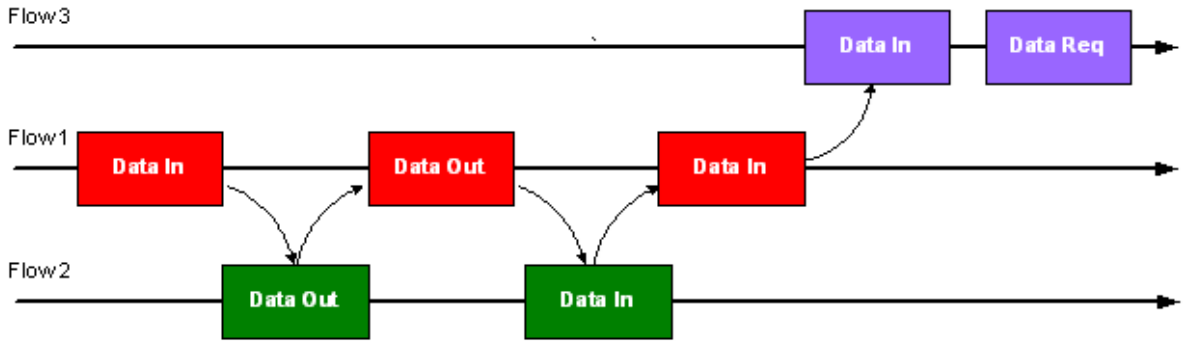

Рисунок 4.1– Транзакційна лінія потоку

4.2 Транзакційний аналіз у Data Mining

Обмін даними - це науковий підхід, що забезпечує ефективні інструменти для отримання знань або розкриття цікавих шаблонів, прихованих у великій кількості даних [24].

Більшість методів обміну даними намагаються зафіксувати деякі локальні залежності залежно від змінних, щоб представити різні аспекти об'єкта статистичним або його еквівалентним методом. Оскільки обсяг даних великий, важко проаналізувати та витягнути з нього інформацію чи знання за допомогою стандартних методик.

Обмін даними може застосовуватися у всіх видах баз даних, таких як реляційні бази даних, дані транзакцій або навіть плоскі файли. Якщо говорити про базу даних, то припустімо, що це предметний, інтегрований, часовий варіант та енергонезалежний збір даних. Він спрямований на об'єднання великого обсягу даних, зібраних з декількох джерел або протягом тривалих періодів часу, у добре організовану узагальнену модель даних. Побудова бази даних означає очищення, перетворення даних та інтеграцію. В основному це важливий крок попередньої обробки для обміну даними [25].

Методи аналізу транзакцій зазвичай розповсюджуються в ринковому аналізі для цільового маркетингу, визначення моделей закупівель клієнтів,

крос-маркетингового аналізу, інтернет-банкінгу тощо. Методи, що використовуються для виявлення різних правил, зразків та асоціацій у транзакційній базі даних, називаються аналізом зв’язків.

4.3 Огляд аналізу посилань

Аналіз зв’язків - це підхід для обміну даними для дослідження за допомогою виявлення зв’язків між значеннями в транзакційній базі даних [24]. Існує два підходи аналізу зв’язку: виявлення асоціації та виявлення послідовностей. Виявлення асоціації шукає залежність предметів, що з'явились разом в одній події. Виявлення послідовності шукає послідовність елементів, пов’язаних у часі.

Асоціації позначають як A => B, де A називається попередником, а B - наслідком. Поява, з якою певна асоціація з'являється в базі даних, - це кількість транзакцій, що містять цю асоціацію. Також відома як кількість частот або число підтримки асоціації:

$$Sup(A=>B) = Freq(A \& B). \quad (4.1)$$

Низький рівень підтримки може вказувати на те, що конкретна асоціація не є дуже важливою, або може вказувати на наявність поганих даних або незначних.

Відносну частоту зустрічальності предметів та їх комбінацій слід аналізувати при виявленні змістовних правил. Мета - знайти умовну передбачуваність B, задану A.

Іншим терміном цієї умовної передбачуваності є впевненість. Впевненість обчислюється як співвідношення:

$$B(A=>B) = Sup(A \& B) / Sup(A). \quad (4.2)$$

Підвищення - це ще одна міра потужності асоціації і відображає вплив виникнення A на ймовірність виникнення B.

Підвищення розраховується як відношення:

$$Lift(A=>B) = Conf(A=>B) / Sup(B) \quad (4.3)$$

Правила, створені алгоритмами асоціації, створюються під час сортування даних та підрахунку подій. Таким чином можна розраховувати впевненість та підвищення. Здатність робити це ефективно, є відмітною особливістю алгоритмів асоціації, через експоненціальне зростання числа правил. У більшості алгоритмів виходить база даних правил, факторів впевненості та підвищення, яку можна отримати.

Ще одна корисна особливість алгоритмів асоціації - це можливість вказувати ієрархію елементів. Тому важливо вибрати належний рівень агрегації. Ієрархія елементів забезпечує ефективний спосіб контролювати рівень агрегації та грати з різними рівнями.

Правила асоціації або послідовності - це описи відносин у певній базі даних. Немає можливості перевірити здібності отриманих правил. Таким чином, правила є лише припущенням про майбутню поведінку.

Те, як отримані правила можуть бути реалізовані, не так очевидно. Тож останнє слово залишається аналітикам, як правильно застосовувати виявлені шаблони. Тому аналіз та експерименти потрібні, щоб отримати будь-яку користь від правил асоціації.

Цим графічні методи також можуть бути дуже корисними для перегляду структури даних. Для того, щоб зробити представництво ще більш зрозумілим, асоціації можуть бути відображені на проблемній області, щоб підкреслити потенційно більш важливі відносини в системі.

Для того, щоб застосувати цю методику до детермінованого моделювання та верифікації цифрових систем, необхідно додатково використовувати формальні методи перевірки, такі як перевірка

еквівалентності. Оскільки методи обміну даними не можуть дати однозначної відповіді про те, чи є дана модель дійсною, виходячи з існуючих специфікацій. Але вони можуть різко скоротити область верифікаційного аналізу за допомогою виділення основних потоків даних та приховування незначних.

4.4 Метод аналізу посилань для моделювання рівня транзакцій

Намагаючись здобути транзакційне налагодження для величезних баз даних транзакцій, запропоновано новий метод частотної схеми для моделювання рівня транзакцій (TLM-FP) [26]. Метод заснований на алгоритмах зростання FP [27] та FP-дерев [28].

Спочатку існує транзакційна база даних, отримана в результаті моделювання моделі ESL. Щоб зменшити розмір бази даних та перетворити її у зручний формат для вилучення частотних шаблонів, необхідно виконати наступні дії:

- позначати одну транзакцію з одного блоку в інший як окремий елемент з відповідною назвою;
- з неодноразові транзакції, послідовно по черзі в базі даних, до нової єдиної транзакції (рисунок 4.2);

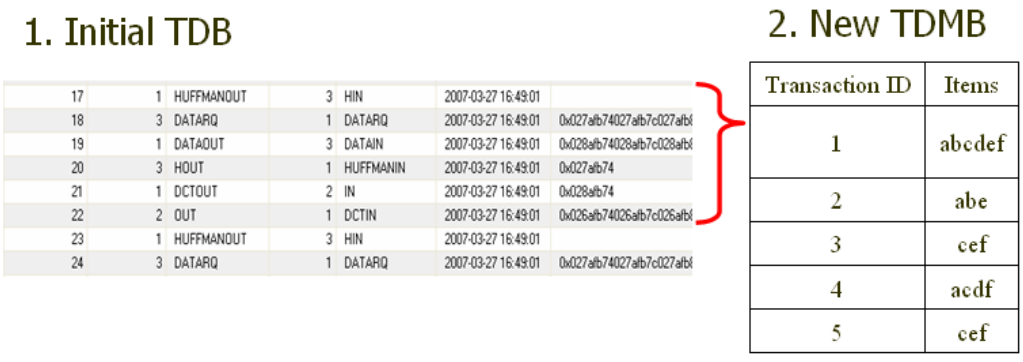

Рисунок 4.2 - Алгоритм TLM-FP: кроки 1-2

Наступні кроки необхідні для побудови дерева FP:

- сканувати нову базу даних один раз, збирати кількість кожного елемента та ліквідувати ті елементи, підтримка яких не переходить визначений поріг підтримки;
- сканувати базу даних транзакцій вдруге; для кожної транзакції відфільтруйте нечасті елементи та відсортуйте решта за порядком у зменшенні частоти.

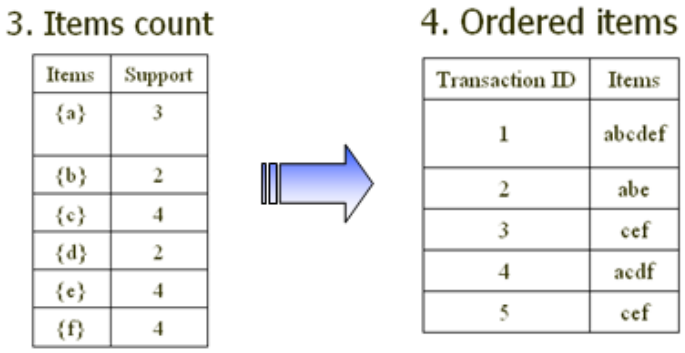

Рисунок 4.3 – Алгоритм TLM-FP: крок 3-4

Вставка у дерево FP як гілка.

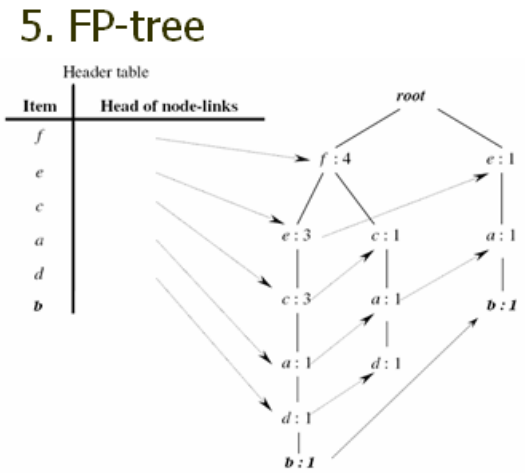

Рисунок 4.4 – Алгоритм TLM-FP: крок 5

Використання алгоритму зростання FP [27] для рекурсивного створення дерева FP для умовних шаблонів (рисунок 4.4):

Модель	Підтримка (>=2)
c	4
e	4
f	4
f , c	4
f , e	3
a	3
f, d	2
f , c, a	2
f , c, a, d	2
f , a, d	2
f , c, d	2
c, d	2
c, a, d	2
d	2

Рисунок 4.5 – TLM-FP алгоритм

Модель	Підтримка (>=2)
c	4
e	4
f	4
f , c	4
f , e	3
a	3
f, d	2
f , c, a	2
f , c, a, d	2
f , a, d	2
f , c, d	2
c, d	2
c, a, d	2
d	2

Рисунок 4.6 – TLM-FP алгоритм

На останньому етапі алгоритм зв‘язку отримуємо з підготовленого дерева FP за допомогою алгоритму зростання FP (рисунок 4.6). Основна ідея методу росту FP - вирощування рекурсивно частих шаблонів за шаблоном та розділом бази даних. Алгоритм складається з наступних етапів:

- Побудувати для кожного частого елемента його умовні баз, а потім його умовне дерево FP;
- Повторити процес на кожному новоствореному умовному FP-дереві;
- Поки порожнє дерево FP не буде порожнім, або воно містить лише один шлях - генеруємо всі комбінації його підпунктів, кожен з яких є частою схемою.

4.5 Результати моделювання рівня транзакцій за допомогою FP дерева

Розглянутий алгоритм дає транзакційну базу даних від симулятора TLM і після застосування декількох отриманих процедур дає часті схеми. Замінні шаблони - це посилання на транзакції або потоки, де кожне посилання (потік) містить набір транзакцій ESL з відповідним рівнем підтримки. Загальний набір посилань можна зменшити відповідно до заданого порогу. Таким чином можна зменшити загальну кількість даних для подальшого аналізу щодо відповідного рівня підтримки. Ці посилання представляють основні цикли даних в системі та рівень підтримки - ступінь популярності транзакцій.

Алгоритм зростання FP, який використовується для видобутку кінцевих умовних моделей, забезпечує хороші показники, пов'язані з відомими аналогами, такими як Apriori та TreeProjection [28]. Таким чином, це призводить до загальної ефективності методу. Для цього потрібне одне сканування бази даних початкового моделювання та лише два сканування трансформованої транзакційної бази даних, де кожна транзакція складається з набору елементів (транзакцій TLM).

Таким чином, нам потрібно сканувати лише таку кількість транзакцій:

$$N = k+2*l\ , \qquad (4.4)$$

де k - кількість записів у початковій базі даних моделювання;

l - ряд транзакцій з трансформованою TDB.

Беручи до уваги, що кількість транзакцій трансформованої бази даних l - у m разів менше, ніж початкової бази даних моделювання, де m - означає середню кількість елементів на транзакцію. Отже вираз (4) можна представити таким чином:

$$N = k(1+2/m)\ , \qquad (4.5)$$

Інші дії необхідні для отримання умовних зразків за допомогою перекидання дерева. Якщо припустити, що середнє значення m дорівнює 5, загальна кількість транзакцій для сканування становить:

$$Nm=5 = 1.4*k, \qquad (4.6)$$

Описаний алгоритм TLM-FP допомагає підтримувати функціональну перевірку моделей ESL. Коли помилка буде визначена під час моделювання, буде виявлений відповідний потік транзакцій. Рухаючись вперед і назад, можна знайти причину помилки. Після цього проблема може бути зміщена до рівня фізичних сигналів.

4.6 Висновок

Частота отримання результату досі ніколи не використовувалася при симуляційному аналізі даних цифрових систем. Запропонований метод потребує лише одного сканування початкової бази даних та 2 сканування трансформованої бази даних. Приблизно потрібно час, рівний 1,5 скануванню в початковій базі даних, порівняно з 2 скануваннями для

застосування оригінального алгоритму FP-дерев для звичайної бази даних транзакцій.

Новий підхід дозволяє представити дані моделювання на системному рівні більш зручним способом транзакційних потоків. У цьому випадку необхідно проаналізувати лише кілька типових транзакційних потоків, а не дивитися на сотні тисяч одиночних транзакцій.

Цей метод економить час під час налагодження. Іншими словами, ця техніка в синтезі з формальними методами перевірки дозволяє швидше перевірити дизайн і отримує кінцеву якість продукції. Використання методу в процесі верифікації буде показано в розділі 6 за допомогою середовища моделювання, описаного в наступному розділі.

5 СЕРЕДОВИЩЕ МОДЕЛЮВАННЯ

5.1 Моделювання архітектури середовища

У цій главі розглядається архітектура експериментального інтегрованого середовища верифікації (IVE). Розроблена система спрямована на швидке проектування та перевірку на системному рівні абстракції моделі. Показані основні компоненти та потоки даних.

Система дозволяє моделювати моделі високого рівня, написані також на C / C ++, як низькорівневі RTL IP-блоки, підключені до системи SystemC Transactor Wrappers, створені за допомогою автоматизованого інструменту. Незважаючи на те, що дане рішення засноване на Aldec® Riviera ™ Verification Solution, система не є тренажером і будь-який інструмент EDA зі здатністю імітувати код SystemC може бути використаний, навіть декілька одночасно в одному проекті.

Основною перевагою системи є можливість зберігання результатів моделювання в транзакційній базі даних з подальшою обробкою та аналізом даних. Архітектура системи моделювання показана на рисунку 5.1

Основна частина IVE - це сервер, який керує потоками даних, передаючи систему. Він складається з поворотної системи, яка перенаправляє трафік від ініціатора до замовника. Спеціальний протокол зв'язку існує для того, щоб це стало можливим. Ще одна важлива особливість - управління дизайном, метою якого є створення моделей IP в бібліотеці дизайнів на основі вхідних інтерфейсів XML.

.

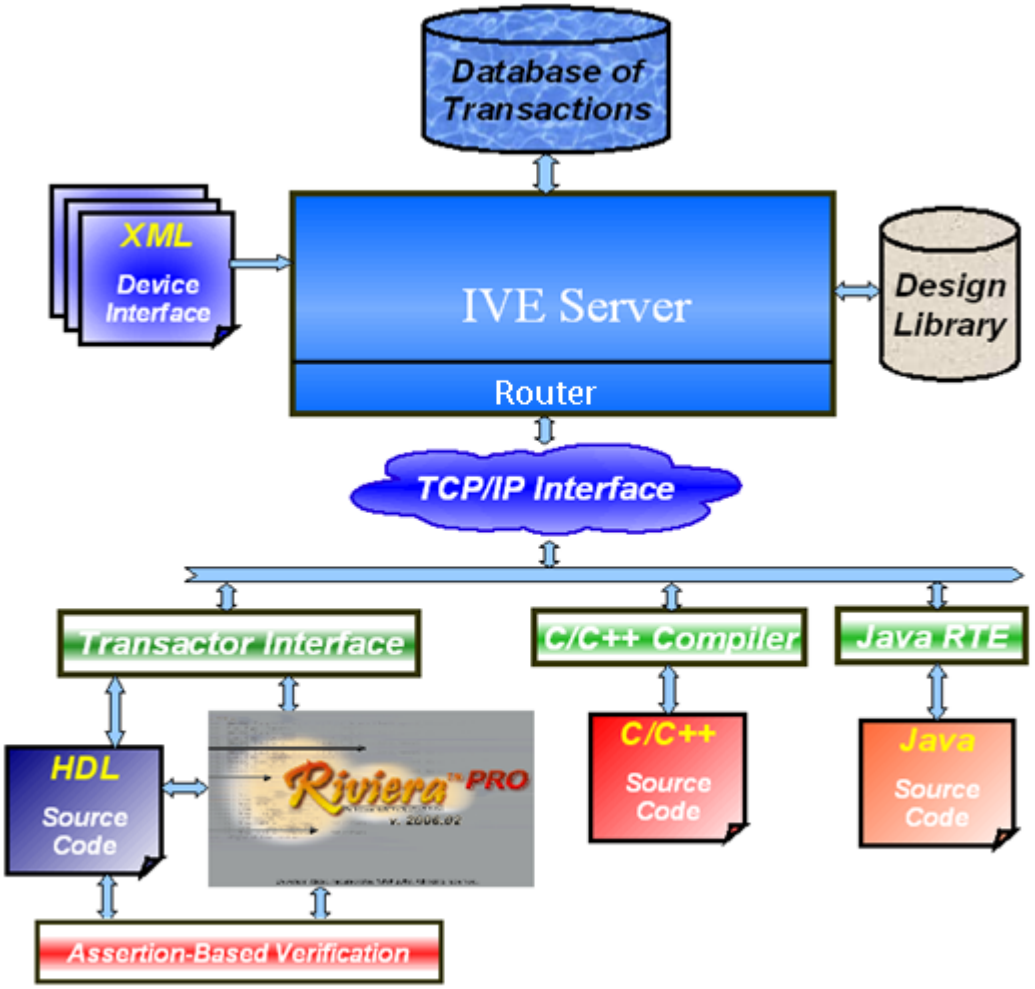

Рисунок 5.1 – Моделювання архітектури системи

Коли новий компонент підключено до проекту, буде створено відповідний зразок моделі. Після створення прототипу він буде пов’язаний з реальною моделлю під час процедури з'єднання.

Середовище зв'язку передачі даних на основі мережевого протоколу TCP/IP. Реалізація передачі пакетів підтримується бібліотекою Windows Socket 2. Усі передані дані зберігаються в спеціальній транзакційній базі даних для подальшого транзакційного аналізу та розкриття порушень обраних параметрів моделі.

За допомогою описаного інтерфейсу зв'язку, що надається спеціальним програмним модулем IVE, клієнтські моделі можуть підключитися до

поточного проекту. Це можуть бути RTL-моделі, запущені симулятором HDL з інтерфейсом Transactor, а також об'єктно-орієнтовані тестові дошки, написані на C / C ++, навіть на Java.

5.2 Рішення для перевірки «Riviera»

Як було зазначено вище, IVE використовує рішення для перевірки Aldec® Riviera ™. Комплект Riviera включає наступні модулі [29]:

- імітаційний двигун SystemC (стандарт IEEE 1666-2006 ™);
- компілятор VHDL (стандарт IEEE 1076-1993 ™);
- компілятор Verilog і SystemVerilog (IEEE 1364-1995 ™, IEEE 1364-2001 ™, IEEE 1800-2005 ™ стандарти);
- компілятори PSL / OVA / SVA;
- компілятор EDIF;
- двигун моделювання HDL / Assertion з командним інтерпретатором (Do і Tcl) та інтерфейсами C / C ++ (PLI / VPI / VHPI);
- двигуни покриття (кодове покриття, покриття гілок, охоплення вираженнями, перемикання вимикачів) та Design Profiler;
- вдосконалені засоби налагодження та різні погляди;
- Lint для VHDL і Verilog (вбудований у компілятори) та двигун ALINT для Verilog.

Ще однією дуже корисною рисою Riviera є майстер транзакцій [30]. Це допомагає написати спеціальне покриття, що дозволяє використовувати моделі HDL на рівні транзакцій. Майстер генерує файли SystemC, які можуть бути використані як відправна точка для розробки тестового стенда для транзакцій. Транзактори зазвичай реалізуються у вигляді набору з трьох класів: класу інтерфейсу сигналу, класу інтерфейсу завдань та класу транзакторів (рисунок 5.2).

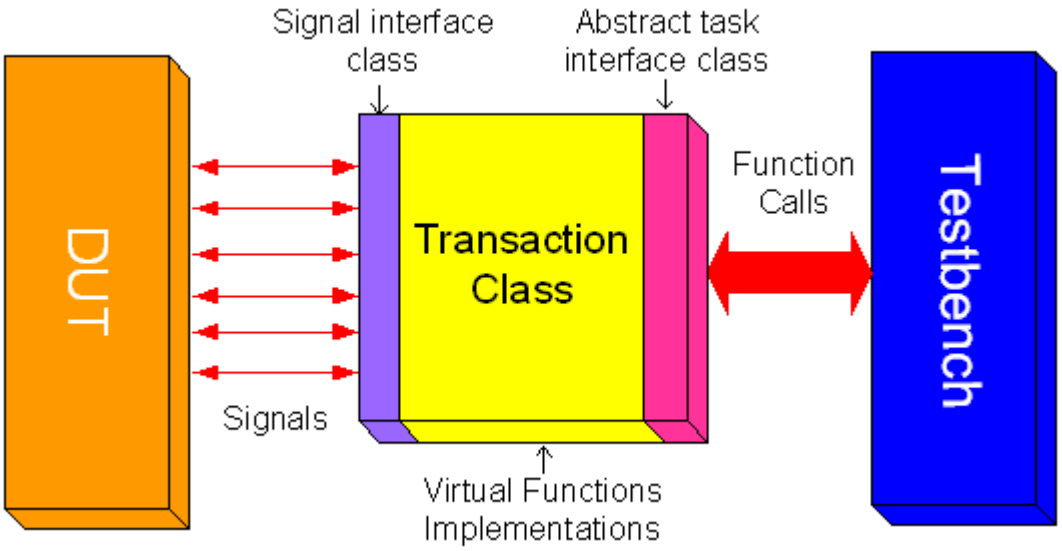

Рисунок 5.2 – Реалізація транзакторів

Клас інтерфейсу сигналу визначає інтерфейс pin-рівня, який використовується для зв'язку з блоками, описаними на рівні RT. Клас інтерфейсу завдань використовується для інтерфейсу високого рівня з тестовою панеллю. Клас транзакторів є похідним від класу інтерфейсу сигналу та класу транзактора.

Згенерований Transactor Wizard обгорткою для моделі HDL завантажується в бібліотеку моделювання поточного проекту RTL в апаратному симуляторі. Таким чином, інтерфейс високого рівня до моделі був готовий для вставки цього компонента в розроблену платформу ESL. Таким чином можна здійснити динамічну перевірку проекту в режимі реального часу, пересилаючи дані з тестових стендів високого рівня в модель RTL, написану на VHDL або Verilog. Крім того, спеціальні мови перевірки, такі як SVA, OVA, PSL, можуть використовуватися в рішенні Riviera для надання підтвердження на основі верифікації. В результаті запропонований підхід до верифікації має інтегрований характер. Функціональна перевірка підтримується на двох рівнях проектування:

- перевірка на основі транзакцій (рівень TLM);
- перевірка на основі твердження (рівень RTL).

5.3 API моделювання

Відповідно до розподіленого характеру системи моделювання можна паралельно зробити процес проектування. Це досить зручно для величезних конструкцій SoC. Після створення віртуального прототипу проекту інші реалізації можуть виконуватися різними інженерними відділами. Особливо, коли дизайн включає вбудоване програмне забезпечення.

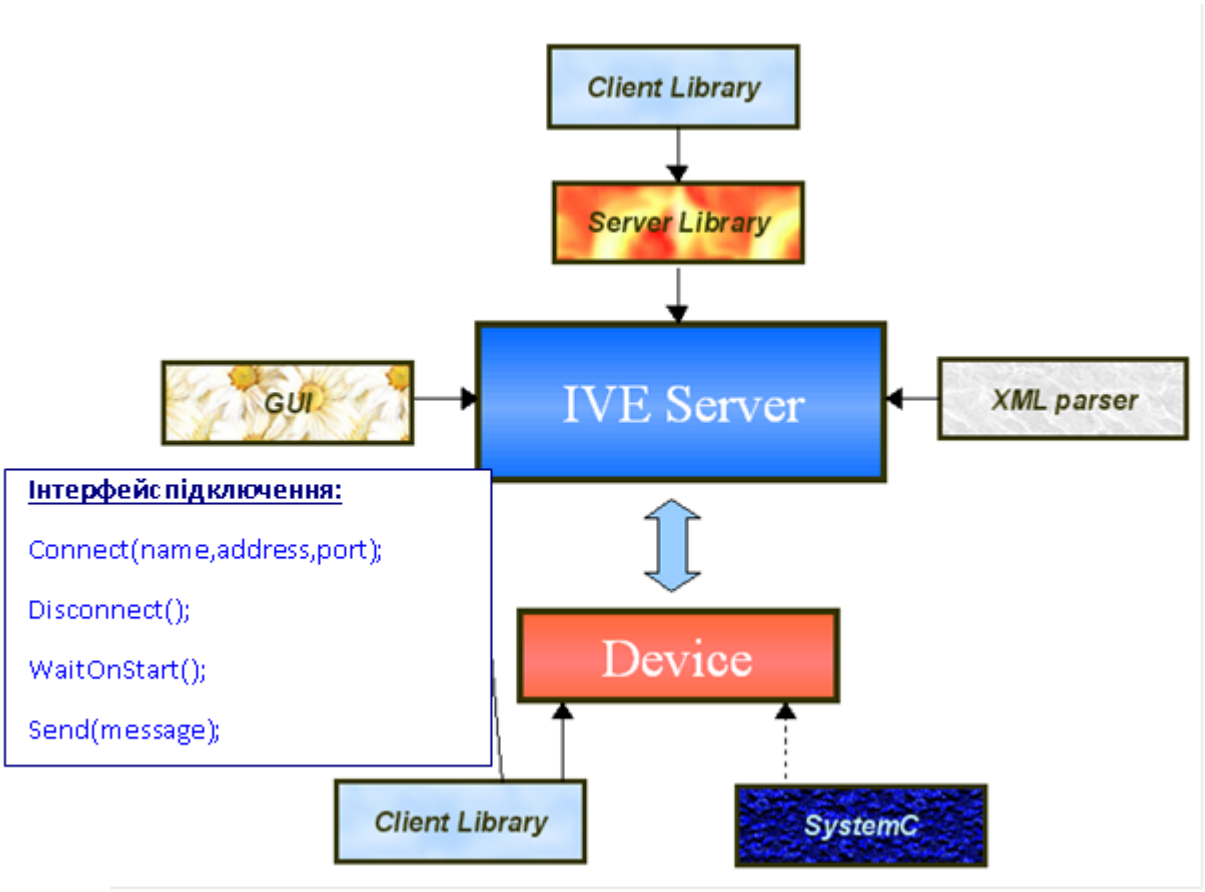

Рисунок 5.3 – Основна структура системного модуля

Основна структура системних модулів показана на рисунку 5.3, що представляє серверно-клієнтську модель системи. Сервер складається з наступних модулів:

- серверна бібліотека - містить функціонал маршрутизації трафіку та менеджер компонентів;

- XML-аналізатор - завантажує блоки пристроїв інтерфейсів;
- GUI - графічний інтерфейс користувача IVE.

Клієнт включає:

- клієнтська бібліотека - підтримує клієнтські компоненти з мережевим зв’язком API для передачі пакетів даних;
- бібліотека SystemC - використовується транзакторами для побудови класів обгортки SystemC моделей HDL.

API клієнта пропонує такі методи:

- Connect (name, address, port) - функція підключення до системного маршрутизатора; Параметри: ім'я пристрою, адреса маршрутизатора та номер порту.
- Disconnect () - відключення пристрою від процесу моделювання.
- WaitOnStart () - метод синхронізації, повертає справжнє значення, коли сервер ініціює процес моделювання.
- Send (message) - надіслати дані моделювання.
- re_message Receive () - отримання даних моделювання.
- CreateMessage (dest, src, data, size, buf) - створити повідомлення із заданими даними для подальшої передачі; Параметри: адреса пристрою призначення, адреса відправника, вказівник на дані, розмір переданих даних, буфер пам'яті, призначений для зберігання повідомлення.
- GetTranactionMessage (header, buf) - витяг заголовка повідомлення із зазначеного буфера пам'яті.
- GetTranactionMessageData (data, buf) - витягує дані повідомлення із зазначеного буфера пам'яті відповідно до інформації заголовка.
- Повідомлення транзакції складається з адресу джерела та місця призначення, самих даних транзакцій, розміру даних та загального розміру пакету (рисунок5.4).

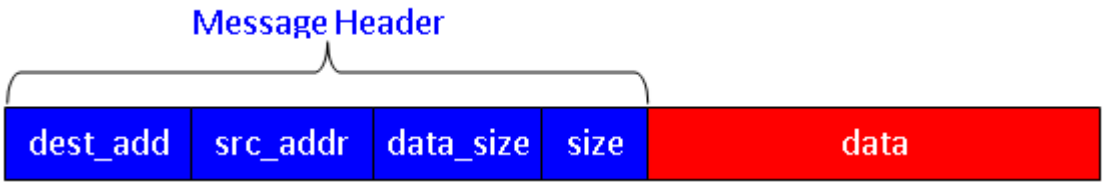

Рисунок 5.4 – Структура повідомлення транзакцій

Описаний інтерфейс зв'язку не вимагає спеціальних знань у передачі мережевих даних. Єдине, що потрібно - це пов’язати клієнтську бібліотеку (так само, як бібліотека SystemC) з перевіреним компонентом, який може бути представлений у моделях транзакторів, а також у тестових групах C / C ++. Після цього згаданий вище інтерфейс доступний для використання.

Представлений API дозволяє моделювати складні системи SoC, щоб забезпечити перевірку на рівні транзакцій. Зразок формальної перевірки джерел різного характеру показаний на рисунку 5.5. Незважаючи на переваги розподілу мережевих комунікаційних підходів, він не може бути досить швидким, щоб задовольнити моделювання з високим насиченням даних. Ось чому незабаром з'являється додатковий режим локального моделювання, де весь трафік буде проходити через спеціальні канали пам'яті.

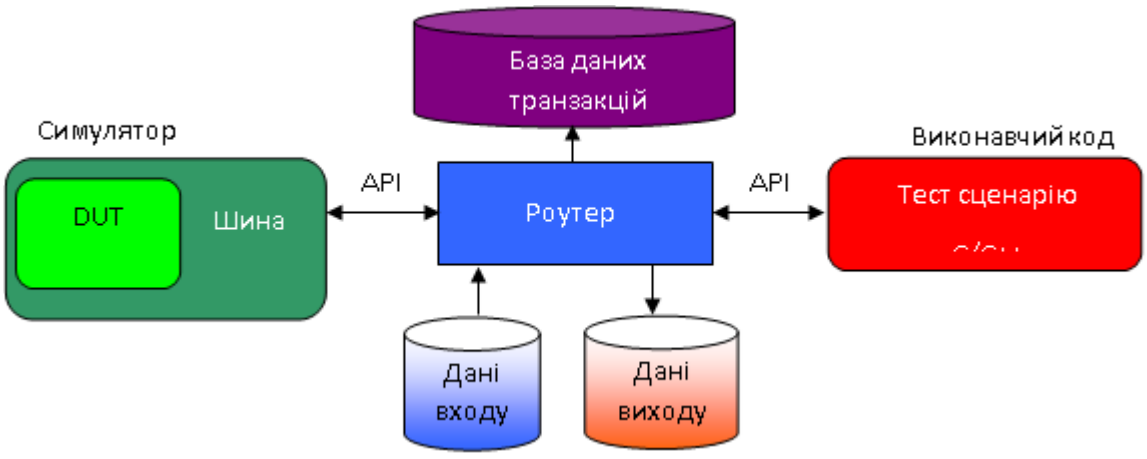

Рисунок 5.5 – Процес верифікації у проекті IVE

5.4 XML-інтерфейс для IP-адреса

Проблема включення блоків SoC може бути вирішена за допомогою створення універсальної мови специфікації інтерфейсу. Для цього створено стандарт SPIRIT (Структура упаковки, інтеграції та повторного використання IP в потоках інструментів) для забезпечення інтеграції IP-блоків із зазначенням вимог до конфігурації для автоматизованих інструментів генерації IP [31].

Це значно мінімізує зусилля зі збирання моделей SoC для архітекторів, дизайнерів, інженерів функціональної перевірки та вбудованих розробників програмного забезпечення. Консорціум SPIRIT розробив стандартний механізм опису та обробки IP-адрес з метою прискорення масштабних проектів SoC шляхом автоматизованої конфігурації та інтеграції конструкцій [32].

В даний час проект IVE не підтримує стандарт SPIRIT, але є можливість реалізувати його в наступній версії системи.

Інтерфейс IP-блоків може бути доданий до середовища моделювання у форматі XML (eXtensible Markup Language). За допомогою завантаження XML-файлу в системі створюється прототип компонента. Метадані поточного формату XML включають лише інформацію про інтерфейси шини, набори вихідних файлів та деяку ієрархічну інформацію. Тим не менш, це допомагає автоматизувати відповідне завантаження компонентів різними джерелами заданого рівня абстрагування. Таким чином інтерфейс пристрою може бути представлений незалежно від мови вихідного коду та формату постачальника. Таким чином, процес налаштування платформи SoC може бути автоматизований, що спрощує процедуру додавання нових IP-адрес.

Розглянемо приклад інтерфейсу XML для компонента пам'яті (рис. 5.6). Тут використовується ієрархія тегів, де ім'я модуля позначено тегом <ім'я>, шлях до вихідного файлу позначений тегом <шлях>. У розділі <port> вказується список каналів та їх цільових з'єднань на рівні транзакцій.

Наприклад, порт «READ» підключений до каналу «IN» модуля контролера. Аналогічно, порт «WRITE» орієнтується на канал «OUT» контролера.

Під час моделювання кожна передана транзакція перевіряється маршрутизатором відповідно до заданого інтерфейсу XML. Якщо виникає помилка, ядро моделювання зупинить процес моделювання із зазначенням неправильної адреси призначення.

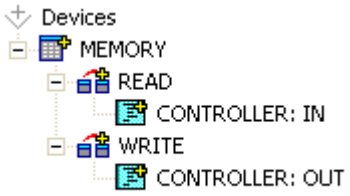

Рисунок 5.6 – XML формат пам‘яті

Після завантаження інтерфейсу пристрою в систему моделювання перелік пристроїв буде доповнений цим компонентом (рисунок 5.7).

```
- <device>
    <name>MEMORY</name>
    <path>"D:\IPS\MemoryRTL\run_test.bat"</path>
  - <port>
      <name>READ</name>
    - <connection>
        <devicename>CONTROLLER</devicename>
        <portname>IN</portname>
      </connection>
    </port>
  - <port>
      <name>WRITE</name>
    - <connection>
        <devicename>CONTROLLER</devicename>
        <portname>OUT</portname>
      </connection>
    </port>
  </device>
```

Рисунок 5.7 – Перегляд інтерфейсу пам'яті

Отже, розроблений XML-інтерфейс забезпечує простий спосіб додавання IP-блоків до проекту SoC, що має гнучкий та простий формат одночасно.

5.5 Протокол передачі даних

Взаємозв'язок об'єктів у системі здійснюється за допомогою передачі повідомлень засобами TCP / IP мережевого протоколу. Реалізація програмного забезпечення базується на рівні Windows Sockets 2.

Windows Sockets 2 (Winsock) дає програмістам можливість створювати мережеві програми для того, щоб передавати дані незалежно від використовуваного мережевого протоколу. За допомогою Winsock можна отримати доступ до розширених мережевих функцій Microsoft Windows, таких як трасляція та якість обслуговування (QoS).

Winsock пропонує модель Windows Open System Architecture (WOSA) зі стандартним сервісним інтерфейсом між API та стеком протоколів, використовуючи парадигму сокета, популяризовану програмою Berkeley Software Distribution (BSD) UNIX. Пізніше він був прийнятий для Windows. Спочатку Winsock орієнтувався на роботу з TCP / IP, а інші протоколи були додані трохи пізніше в Windows Sockets 2.

Для зручності роботи з Windows Sockets був розроблений спеціальний клас обгортки. На відміну від аналога CSocket MFC (Microsoft Foundation Classes), цей клас не використовує механізм обміну повідомленнями Windows. Це обмеження застосовується внаслідок архітектурних особливостей системи моделювання. Необхідно створити об’єкт вікна, щоб мати справу з пулом повідомлень. Однак клієнтами системи є консольні програми, включаючи транзактори SystemC.

В якості рішення були запропоновані багатопотокові приймачі. Зі створенням об'єктів сокета паралельні потоки відновлюються для приймання вхідних пакетів даних. Коли повідомлення надійде, для виклику транзакції

буде викликана зареєстрована функція зворотного виклику зазначеного компонента.

Коли сервер запускається, інший потік приймання відкривається для виявлення підключень клієнтів до певного порту. Орієнтовна схема взаємозв'язку об'єктів показана на діаграмі UML (Unified Modeling Language) (рисунок 5.8):

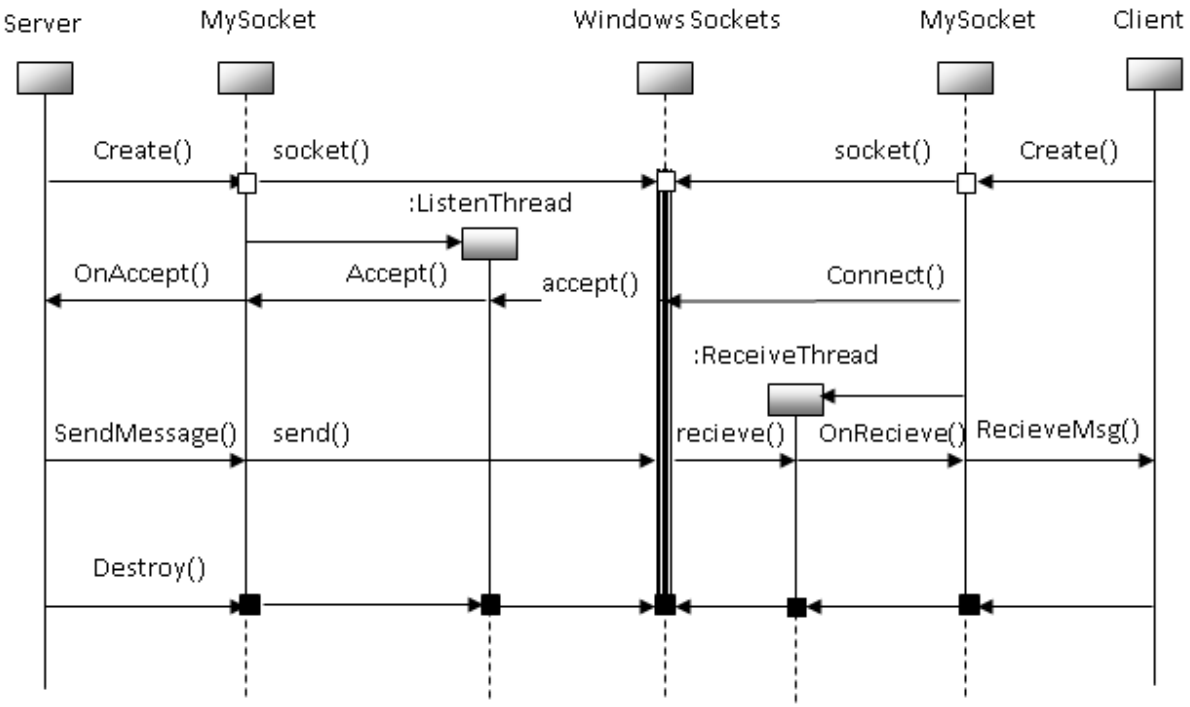

Рисунок 5.8 – Схема UML встановлення з'єднання (клієнт-сервер)

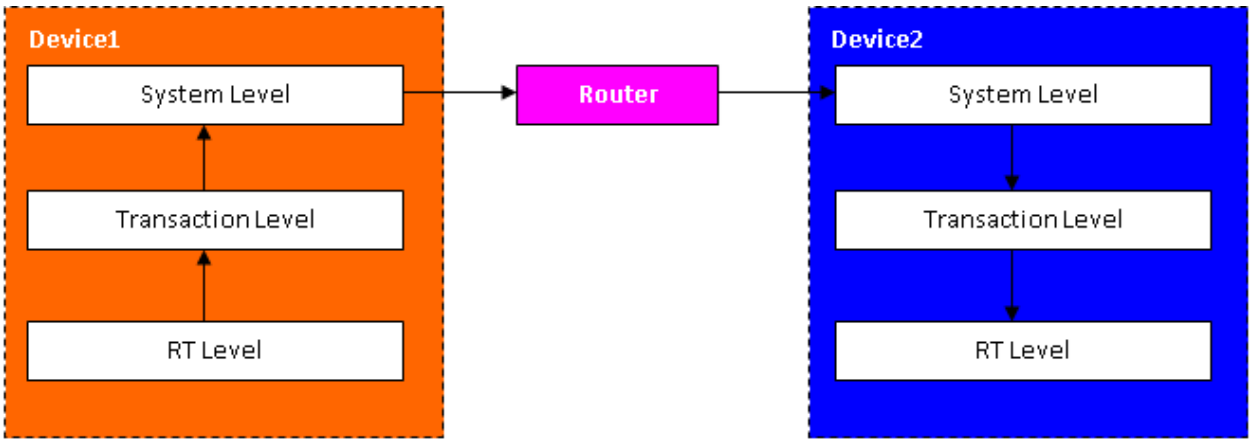

Рисунок 5.9 – З'єднання пристроїв

Шлях даних двох моделей RTL в системі зображено на рисунку 5.9. Для виконання моделювання системного рівня необхідно перетворити дані з рівня сигналів моделі RTL на системний рівень. Це досягається вставкою транзакційних об'єктів у потік даних. Таким чином, дані можуть бути представлені однією транзакцією і потім передані як маршрутизатор системного рівня без надмірного опису сигналів.

Враховуючи, що моделювання відбувається на системному рівні, де компоненти представлені прототипами об'єктів із заздалегідь заданою функціональністю, можуть застосовуватися об'єктно-орієнтовані принципи. Об'єкт представлений інтерфейсом зв'язку, але реалізація цього інтерфейсу лежить всередині об'єктів рівня транзакцій. Таким чином, не потрібно думати про конкретну реалізацію, просто функція використовується відповідно до заданого інтерфейсу.

Процес передачі даних виділено на малюнку 5.10. Початкові дані додаються заголовком транзакції, де вказана адреса призначення. Після додавання спеціальної інформаційної системи з мережевими параметрами готовий пакет надсилається до цільового пристрою, кидаючи маршрутизатор за допомогою мережевого протоколу TCP / IP.

Коли транзакція надходить до маршрутизатора, ім'я компонента призначення витягується із заголовка транзакції та шукається у спеціальній хеш-таблиці. За назвою об'єкта може бути отримана відповідна інформація про нього, наприклад інтерфейс пристрою, опорний сокет, який використовується для з'єднання з процесом пристрою.

Після того, як пакет отриманий клієнтом, ім'я порту аналізується і відповідно до його значення транзактор надсилає вхідні дані в асоційований модуль RTL, який працює під симулятором HDL.

Якщо кінцевим є тестовий додаток, надіслані дані будуть перевірені на правильність і буде прийнято відповідне рішення.

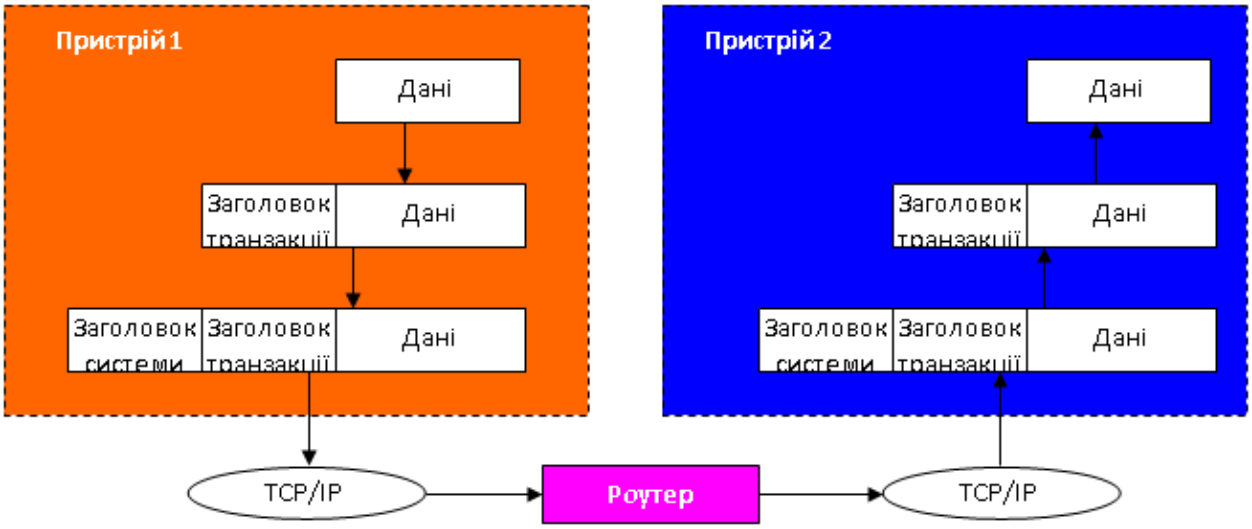

Рисунок 5.10 – Процес передачі даних між пристроями

5.6 Об'єктна модель ядра імітаційного моделювання

Основою будь-якого інструменту компіляції та моделювання є внутрішня семантична модель - ієрархія структури даних, що описує елементи та внутрішні принципи області знань. Гнучка ієрархія структури даних є необхідною умовою моделювання якості та продуктивності системи. Крім того, це взагалі впливає на трудомісткість і подальше обслуговування.

Розроблена система верифікації являє собою багатофункціональну систему з вдосконаленим інтерфейсом для підключення модуля. Цей інтерфейс використовується і клієнтською програмою, як сервер. Для цього була побудована спеціальна ієрархія класів. У нашому випадку система складається з розподілених елементів. Кожен з них має унікальне ім’я та параметри мережі: IP-адрес та номер порту.

Схема на рисунку 5.11 представляє ієрархію класів зв'язку:

- клас Communicator - забезпечує загальний інтерфейс зв'язку на основі функціональності сокетів, що дозволяє підключатись та передавати дані між об'єктами;
- клас ClientCom – обслуговує клієнтський додаток при передачі даних на сервер і назад;

- клас ServerCom - керує з'єднаннями, коли клієнт намагається встановити з'єднання з сервером, клас додає його до спеціального списку з'єднань.

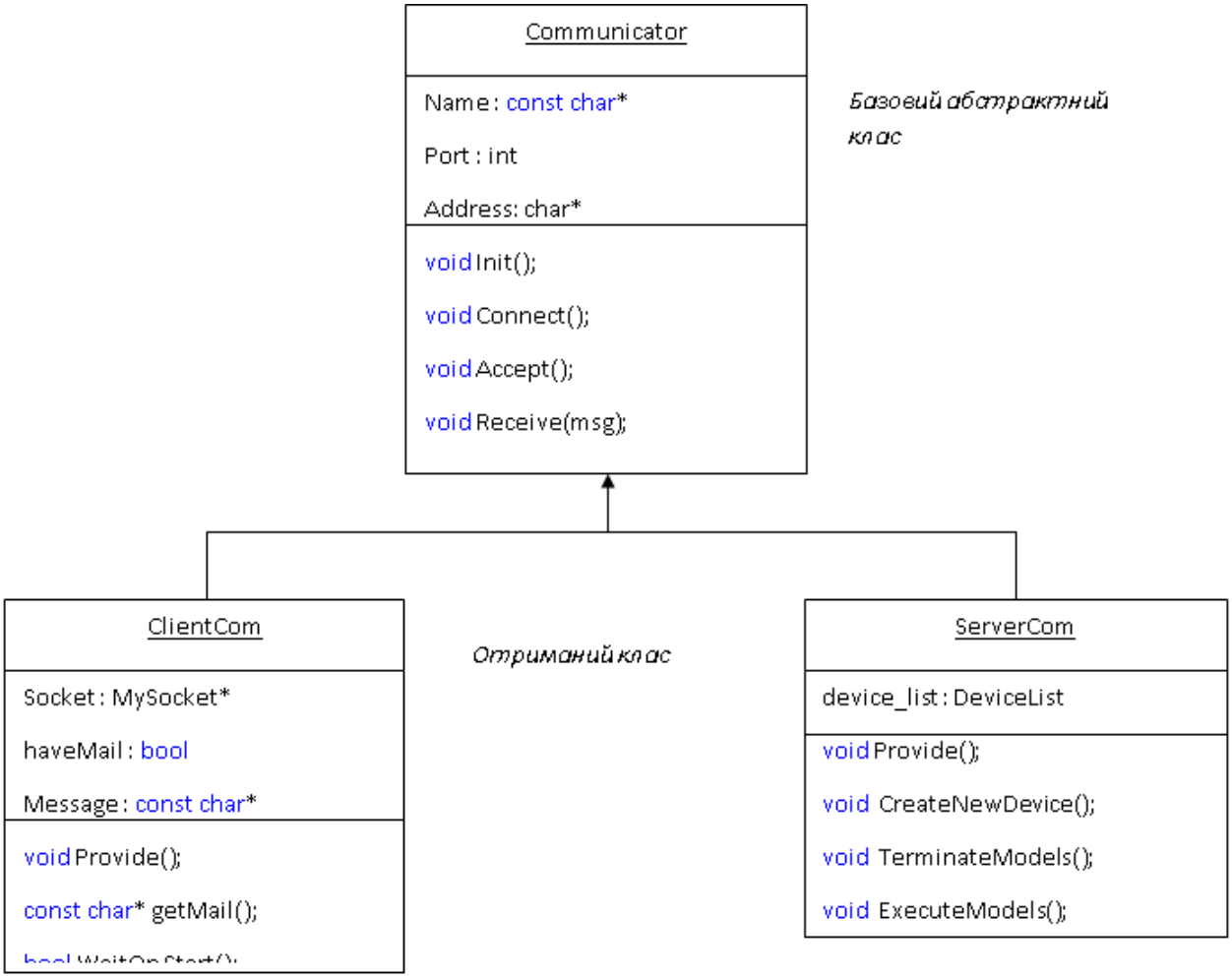

Рисунок 5.11 – Ієрархія класів комунікацій

Намагаючись спростити взаємозв’язок інформації, що надходить з різних системних сторін, та інкапсулювати її, був розроблений спеціальний клас. Об'єктом цього класу є віртуальне представлення пристрою в системі (рисунок 5.12).

Кожному об'єкту присвоюється ім'я, сокет та інформація про інтерфейс пристрою, завантажений у форматі XML. З відкриттям специфікації XML в системі створюється новий об'єкт.

У об'єктній моделі є наступні класи:

- клас DeviceObj - містить загальну інформацію про пристрій, необхідну для моделювання процесу: ім'я, інформацію про інтерфейс, параметри з'єднання;

- клас DeviceInfo - надає початкову інформацію для створення об'єкта DeviceObj; він складається з даних, отриманих після розбору файлів XML файлів пристрою, таких як назва пристрою, список портів, шлях до програми-джерела, обробка процесу;

- порт класу - абстрактний порт пристрою на рівні транзакцій, включаючи цільове з'єднання;

- структура Connection - містить назви пристроїв та портів для поточного з'єднання.

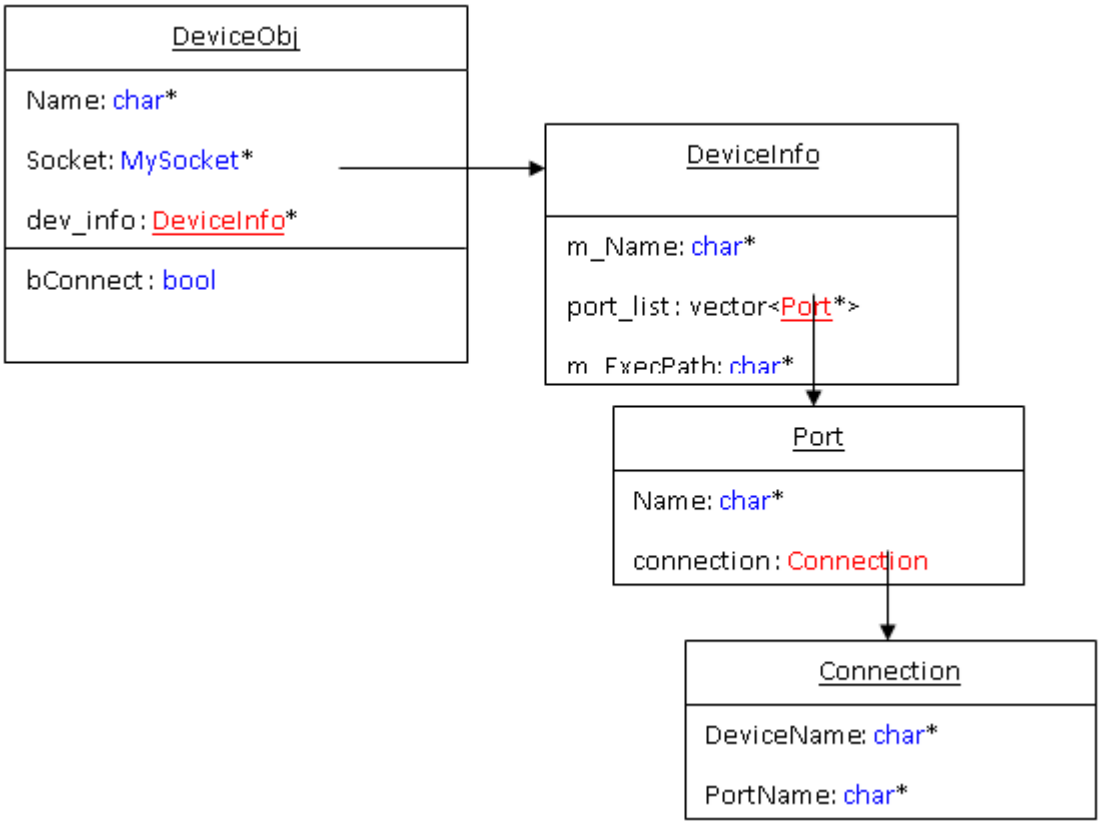

Рисунок 5.12 – Ієрархія класів пристроїв

5.7 Основні характеристики системи

Незважаючи на експериментальний характер, розроблена програмна система дозволяє виконувати основні функціональні можливості проектування та перевірки. У цій версії доступні наступні системні функції:

- спільне моделювання компонентів, описаних на різному рівні абстракції;
- моделювання транзакторів за допомогою симулятора HDL (підтримка SystemC є обов'язковою);
- блоки взаємозв'язку кидають мережевий протокол;
- введення нових компонентів у середовище моделювання у форматі XML;
- запис транзакцій;
- розширений плагін для транзакційного аналізу.

Це дає можливість вирішувати наступні проблеми:

- віртуальне прототипування моделі SoC;
- дострокова перевірка розроблених компонентів HW / SW;
- аналіз комбінацій між HW/SW;
- архітектурна розвідка;
- перевірка на рівні транзакцій;
- транзакційні налагодження за допомогою нового алгоритму аналізу посилань.

5.8 Висновок

Підводячи підсумок, представлене рішення робить дизайн та верифікацію складних проектів SoC (більше одного мільйона виходів) швидшими, починаючи процес розробки у високому рівні абстракції (ESL) та надаючи всі написані вище методи, такі як TLM, XML-інтерфейс для IP-блоків, що включають, записують транзакції в базу даних та можливість

включення плагінів аналізу транзакцій.

Як результат, система моделювання може використовуватися з будь-яким рішенням EDA, якщо вона підтримує стандарт SystemC 2.0 і легко розширюється за допомогою нових функцій аналізу транзакцій. У цьому робочому середовищі моделювання використовується рішення для верифікації Riviera, яке підтримує моделювання за допомогою модулів SystemC та містить інструмент для генерації транзакторів.

6 JPEG-ДЕКОДЕР ПОТОКУ ДАНИХ

6.1 Опис проекту

JPEG - широко використовуваний метод стиснення зображень. Він використовується в численних системах цифрової обробки зображень, таких як сучасні телефони та цифрові камери. Вони часто потребують високошвидкісної системи стиснення зображень. Для задоволення цієї потреби в цифровому процесорі сигналу використовується додатковий IP-блок, який виконує декодування JPEG.

Для зменшення розміру зображення без великих втрат якості використовується техніка кодування JPEG. Кодер JPEG ділить зображення на блоки розміром 8 на 8 пікселів - мінімально кодовані одиниці (MCU). Після цього кодер застосовує ряд операцій на кожному з цих блоків, які при розміщенні в потрібному порядку формують вихідне зображення. Серед них дискретна косинусна трансформація, зигзагоподібне сканування, квантування та кодування змінної довжини. Результатом виходу є стиснене зображення.

Декодер перевертає перетворення, застосовані кодером до даних зображення. Далі декодер застосовує набір дій над стислими даними зображення, такі як декодування змінної довжини [VLD], зигзагоподібне сканування [ZZ], деквантизація [DQ], обернена дискретна косинусна трансформація [IDCT], перетворення кольорів [YUV] та упорядкування до нього [33]. Потім він отримує реконструйоване зображення (рисунок 6.1).

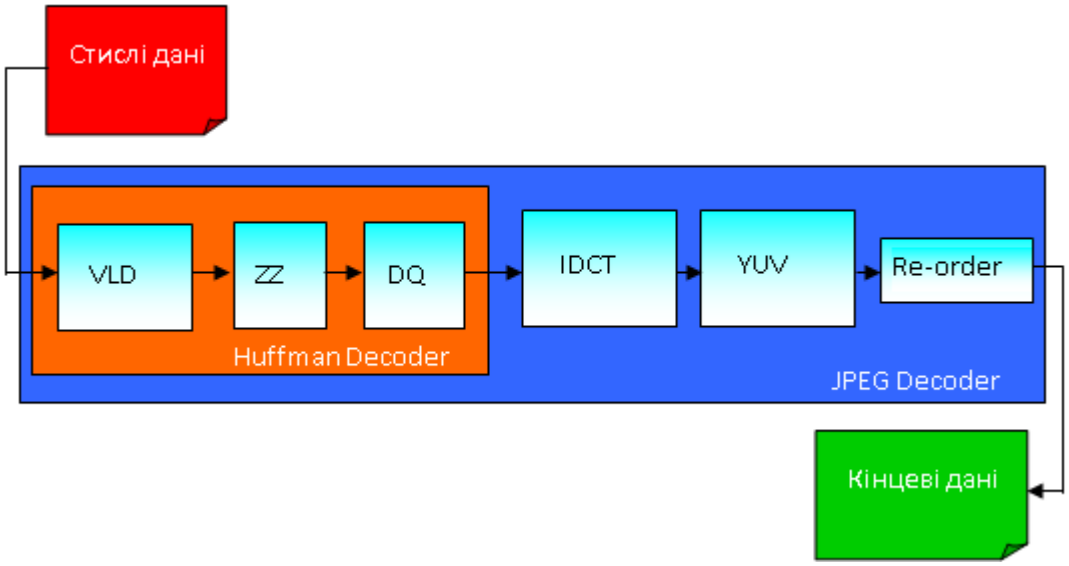

Рисунок 6.1 – JPEG-декодування потоку даних

Віртуальний прототип декодера був створений на основі алгоритму декодування JPEG, записаного на C та завантаженого з [34]. За допомогою проектування прототипу оптимізація може бути виконана ефективно.

Запропонована архітектура прототипу, яка реалізує декодер JPEG. Декодер розділений на кілька блоків обробки: декодер Хаффмана, декодер IDCT, блок перетворення кольорів YUV та контролер даних, який завантажує стислі дані з файлу JPEG і зберігає нестиснуту інформацію RGB у файл результатів BMP.

Для забезпечення зв'язку між різними підсистемами використовується протокол зв'язку системи моделювання. За допомогою цього протоколу MCU може передаватися між підсистемами в процесі декодування.

6.2 Проблеми верифікації

Як було зазначено мета функціональної перевірки - перевірити, чи відповідає поведінкова конструкція функціональним вимогам. Функціональна перевірка та проектування - це інтерактивні процеси, які виконуються до досягнення прийнятного функціонального дизайну.

Для будь-якого проекту SoC важливо розробити та задокументувати план верифікації, який може бути представлений як карта для всіх перевірочних дій, що виконуються.

Цей план визначає процес верифікації і повинен складатися з таких пунктів:

- загальна функціональність дизайну;
- підхід і стратегія верифікації;
- рівень абстракції;
- перевірка результатів;
- джерела моделей та тестових стендів;
- питання управління проектами.

Короткий опис проекту вже наведено в цій главі. Методика системного рівня, про яку йшлося раніше, вибирається як підхід до верифікації на основі моделювання рівня транзакцій.

Схема процесу верифікації показана на рисунку 6.2. Перевірка системного рівня включена на ранніх етапах процесу проектування, таких як віртуальна прототипізація, що включає розробку інтерфейсів компонентів та схему синхронізації прототипів високого рівня.

Результати перевірятимуться виконаною специфікацією декодера за допомогою порівняння виходу моделі, що перевіряється, з виконуваною золотою моделлю декодера.

Після завершення моделювання всі результати моделювання збираються та зберігаються за допомогою моделювання середовища в транзакційній базі даних. Це джерело для подальшого аналізу транзакцій у процесі транзакційного налагодження.

Результати моделювання повинні перевірятись на еквівалентність золотої моделі (тобто виконується специфікація). Якщо вихідна модель не дорівнює очікуваним значенням, перевірка переходить до стадії налагодження транзакційних з метою пошуку помилки в дизайні. На цьому етапі проводиться глибокий транзакційний аналіз з метою пошуку та

виправлення помилки. Після перегляду дизайну слід розпочати новий цикл моделювання.

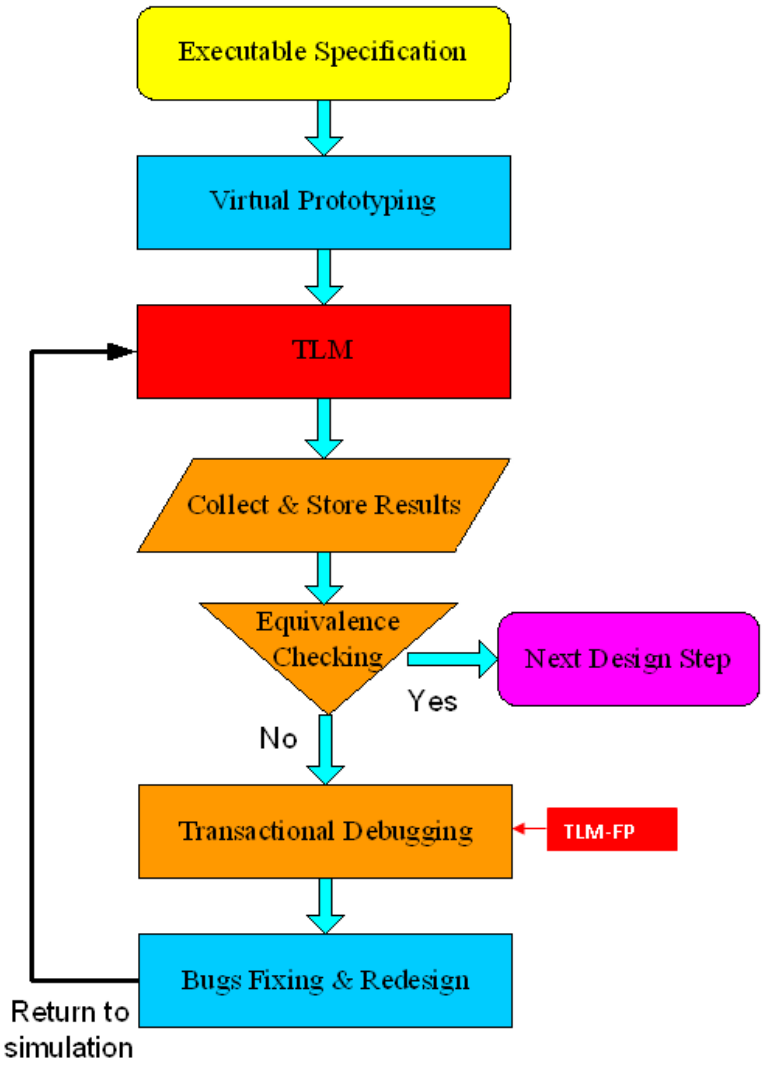

Рисунок 6.2 – Процес верифікації моделі декодера

У випадку, коли перевірка еквівалентності проходить і інші параметри проектування є задовільними, процес переходить до наступного етапу.

6.3 Віртуальний прототип

Після розподілу декодера на кілька підсистем наступним важливим моментом є розробка інтерфейсу зв'язку між розділеними блоками.

Забезпечуючи взаємозв'язок двох блоків, потрібно не лише забезпечувати канали для передачі даних.

Складніше завдання - спроектувати структуру синхронізації системи. Мета - реалізувати мінімально необхідну кількість точок синхронізації. В іншому випадку блоки оброблять помилкові дані внаслідок розладу транзакцій або занадто багато подій синхронізації сповільнять систему.

На рисунку 6.3 представлений кінцевий прототип моделі, який фіксується в інтерфейсах XML. Більше того, кожен блок моделі - це окремий процес з інтерфейсом, реалізованим мовою C.

Цю модель легко використовувати для тестування початкової конфігурації, що базується лише на застосуванні алгоритму декодування із відповідним завантажувачем зображень JPEG та завантажувачем файлів BMP.

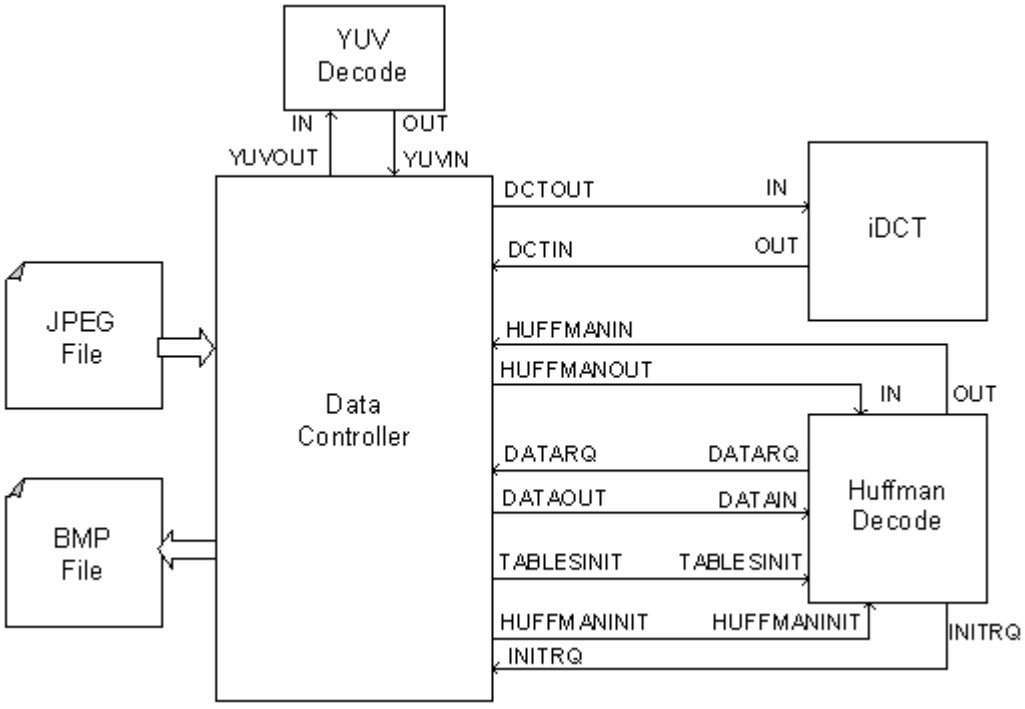

Рисунок 6.3 – прототип JPEG-декодера

6.4 Транзакційний аналіз даних

Коли побудовано віртуальний прототип і виконано моделювання, зібрані результати слід зберігати в базі даних транзакцій, яка є джерелом для подальшого транзакційного аналізу. База даних містить інформацію про кожну транзакцію в системі, включаючи шлях транзакції, додавання часу та самих даних транзакцій (рисунок 6.4).

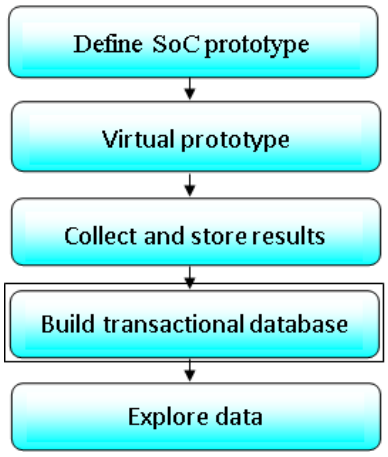

Рисунок 6.4 – База даних транзакцій

transact_id	transact_from	transact_port_from	transact_to	transact_port_to	transact_time	transact_data
1	3	INITRQ	1	INITRQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
2	1	TABLESINIT	3	TABLESINIT	2007-03-27 16:49:01	0x028afb74028afb7c028afb84028afb8c028afb940
3	3	INITRQ	1	INITRQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
4	1	TABLESINIT	3	TABLESINIT	2007-03-27 16:49:01	
5	3	INITRQ	1	INITRQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
6	1	TABLESINIT	3	TABLESINIT	2007-03-27 16:49:01	
7	3	INITRQ	1	INITRQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
8	1	TABLESINIT	3	TABLESINIT	2007-03-27 16:49:01	
9	3	INITRQ	1	INITRQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
10	1	HUFFMANINIT	3	HUFFMANINIT	2007-03-27 16:49:01	0x028afb74
11	1	HUFFMANOUT	3	HIN	2007-03-27 16:49:01	
12	3	DATARQ	1	DATARQ	2007-03-27 16:49:01	0x027afb74027afb7c027afb84027afb8c027afb940
13	1	DATAOUT	3	DATAIN	2007-03-27 16:49:01	0x028afb74028afb7c028afb84028afb8c028afb940
14	3	HOUT	1	HUFFMANIN	2007-03-27 16:49:01	0x027afb74
15	1	DCTOUT	2	IN	2007-03-27 16:49:01	0x028afb74
16	2	OUT	1	DCTIN	2007-03-27 16:49:01	0x026afb74026afb7c026afb84026afb8c026afb940
17	1	HUFFMANOUT	3	HIN	2007-03-27 16:49:01	

Рисунок 6.5 – Аналіз потоку даних

Загальна схема аналізу даних показана на рисунку 6.6, де після моделювання прототипу та збору результатів транзакційної бази даних слід побудувати.

База даних транзакцій, що використовується для зберігання інформації про моделювання, має надлишкові дані, які не потрібні для аналізу посилань після обробки. Отже, спеціальний обмін даними повинен бути створений лише з суттєвою інформацією про транзакційні потоки, що включає інформацію про джерела та блоки призначення та використаний канал даних. Для цього спочатку таблицю кодування необхідно заповнити всіма існуючими типами транзакцій. До кожної такої транзакції пов'язане просте стиснення символів (рисунок 6.5).

Contraction	Transaction Source	Transaction Destination
a	HUFFMAN:INITRQ	CONTROLLER:INITRQ
b	CONTROLLER:TABLESINIT	HUFFMAN:TABLESINIT
c	CONTROLLER:HUFFMANINIT	HUFFMAN:HUFFMANINIT
d	CONTROLLER:HUFFMANOUT	HUFFMAN:HIN
e	HUFFMAN:DATARQ	CONTROLLER:DATARQ
f	CONTROLLER:DATAOUT	HUFFMAN:DATAIN
g	HUFFMAN:HOUT	CONTROLLER:HUFFMANIN
h	CONTROLLER:DCTOUT	iDCT:IN
i	iDCT:OUT	CONTROLLER:DCTIN
j	CONTROLLER:YUVOUT	YUV:IN
k	YUV:OUT	CONTROLLER:YUVIN

Рисунок 6.6 – Таблиця кодування транзакцій

Перші три транзакції використовуються для ініціалізації таблиць DHT, DQT, HT, HN в декодері Хаффмана. Тож вони трапляються лише один раз і не впливають на загальну статистику транзакцій (рисунок 6.7).

Ця статистика представляє процес декодування для зображення 2,07 Мп (1920x1080). Ще одна схема показує виклики пристрою для того ж випадку (рисунок 6.8). Очевидно, що блок Controller керує всіма транзакціями в системі, тому обробляє максимальну кількість транзакцій.

Але ці значення нічого не говорять про продуктивність і не можуть бути використані для визначення реального використання компонента.

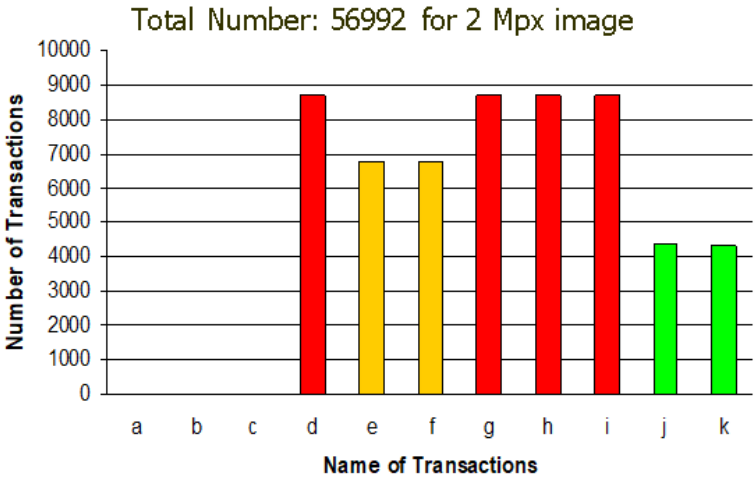

Рисунок 6.7 – Статистика транзакцій

Тому що час обчислень на рівні реалізації невідомий і може змінюватись від транзакції до транзакції з причини поганого моделювання. Тому високий рівень транзакцій для пристрою не означає високого використання. Це означає лише високу ймовірність виявлення помилки на цьому пристрої.

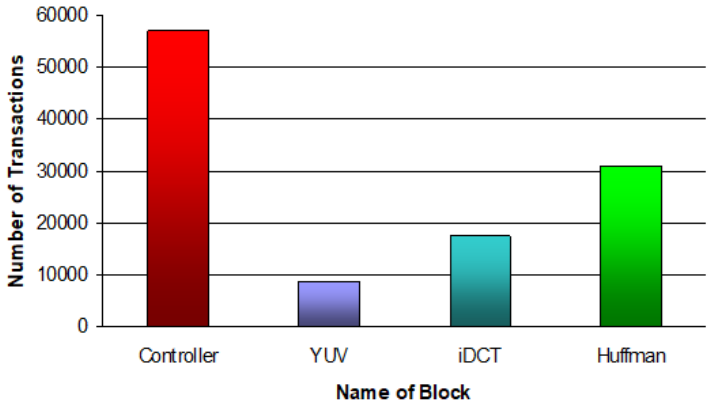

Рисунок 6.8 – Транзакційні виклики пристроїв

Після побудови бази даних даних загальна кількість записів зменшилася в 5 разів з 56 992 до 11 736 транзакцій. Нова база даних складається з нециклічної послідовності транзакцій з імітаційної бази даних. Відповідно до розміру нової бази даних, кожна нова транзакція включає в середньому п'ять старих транзакцій. Ця база даних виглядає зручніше для аналізу посилань (таблиця 6.1).

Наступна операція виключає транзакції, які мають рівень підтримки менший за визначений поріг. У цьому випадку вона дорівнює 0,1, і досить видалити ініціалізаційні операції - a, b, c. Вони не можуть вплинути на результати, але можуть трохи уповільнити процес аналізу.

№	Потік транзакцій
1	ab
2	ab
3	ab
4	acdefghi
5	defghi
6	defghi
7	dghi
8	defghi

Рисунок 6.9 – Потік транзакцій

Після цього алгоритм будує дерево FP та генерує ланцюги транзакцій. Ці ланцюги знову фільтруються і отримуємо остаточні результати (рисунок 6.10).

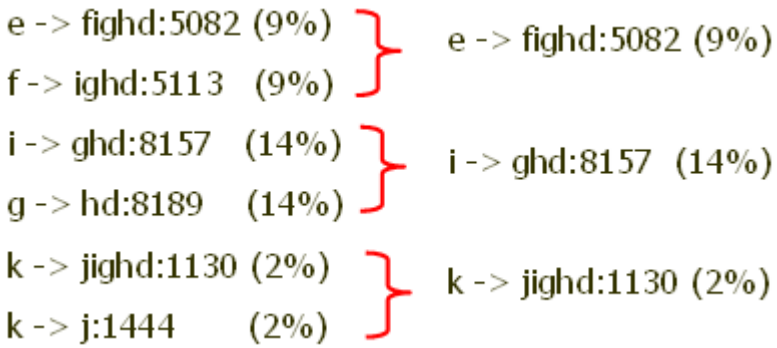

Рисунок 6.10 – Транзакційні зв’язки

Зацікавившись лише найдовшими ланцюгами, можна зменшити набір ланок. В результаті три ланки, показані на рисунку 6.8, представляють основні транзакційні потоки в системі.

Коли помилка відслідковується в якійсь транзакції, відповідно до назви коду, насамперед, слід проаналізувати відповідний транзакційний зв’язок із найвищим рівнем підтримки. Переходячи посилання вперед і назад, легко перевірити повний цикл даних з усіма перетвореннями. Таким чином, можна перевірити не окрему транзакцію, а транзакційний потік на еквівалентність специфікації.

Ці закодовані посилання все ще не такі зрозумілі для інженера. Тому з метою досягнення рівня сприйняття було б корисно відобразити ці результати на схемі віртуального прототипу. Таким чином потоки будуть відображені більш очевидним чином.

Потік з максимальним рівнем підтримки, рівним 14%, присвячений основному циклу декодування з перетворенням Хаффмана та IDCT (рис. 6.9). Ще один із рівнем підтримки рівним 9% додатково показує завантаження нових стислих даних для подальшого декодування (рис. 6.10). Це менш часта робота через специфічний алгоритм.

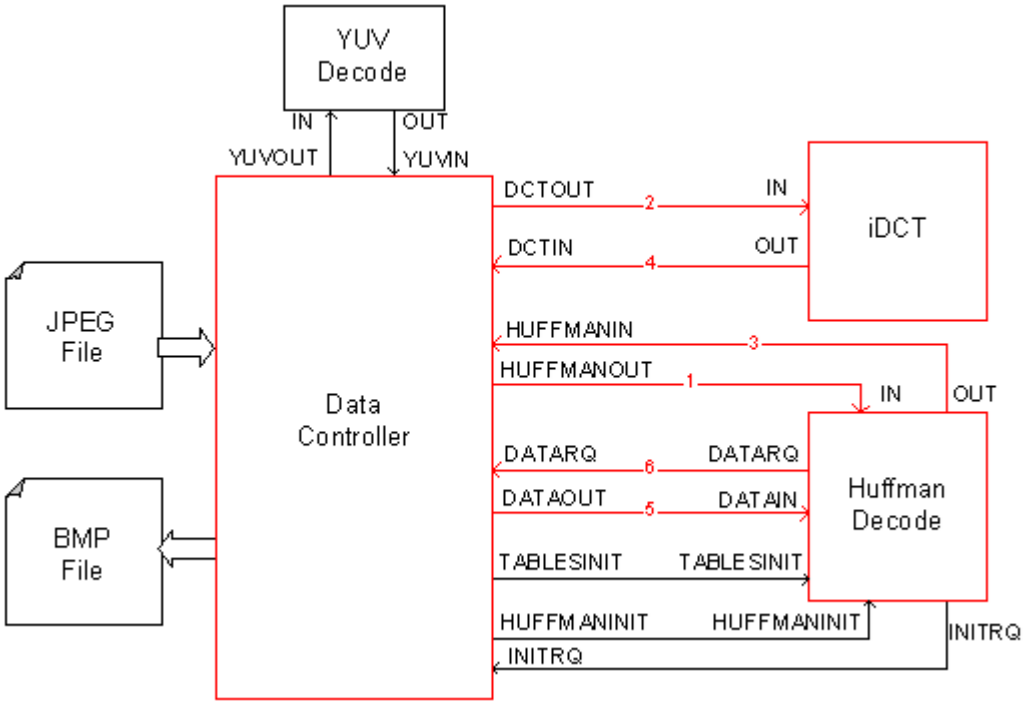

Рисунок 6.11 – Транзакційний потік - ighd: 14%

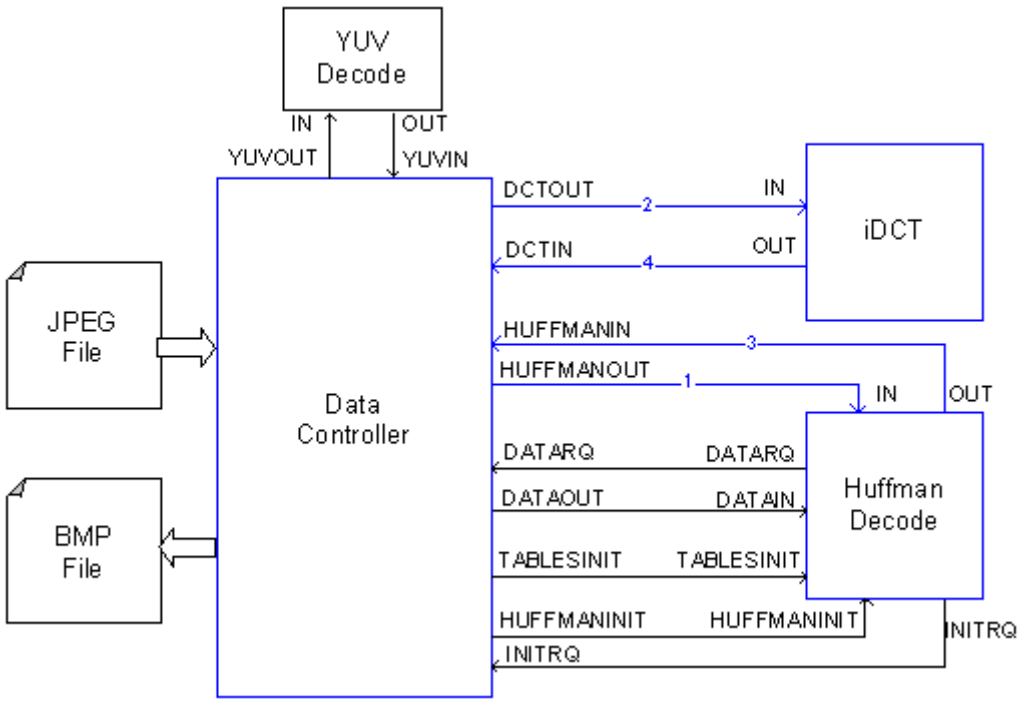

Рисунок 6.12 – Транзакційний потік - efighd: 9%

Останній, але не менш важливий потік транзакцій також включає процедуру перетворення кольорів (рисунок 6.13). Її рівень підтримки дорівнює 2% і підкреслює завершення поточного перетворення піксельного блоку.

Потоки, відфільтровані відповідно до рівня підтримки, стосуються ініціалізації таблиць Хаффмана. Вони з'являються лише один раз і їх можна налагодити вручну.

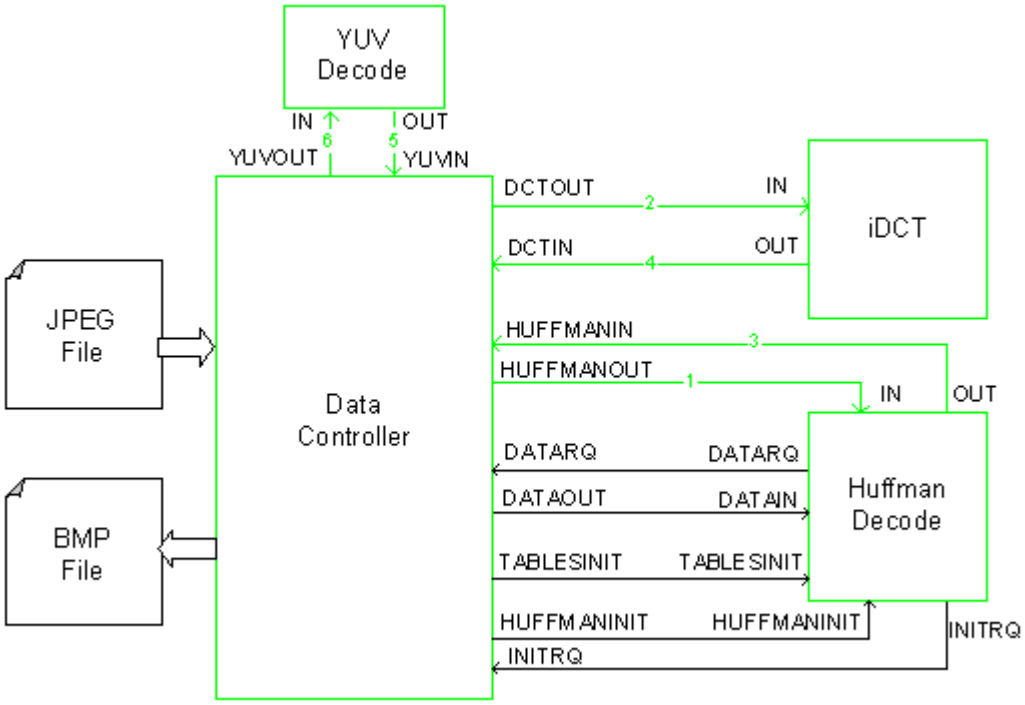

Рисунок 6.13 – Транзакційний потік - kjighd: 2%

6.5 Висновок

Проблема верифікації була побудована на основі простого дизайну SoC. Враховуючи модель декодера JPEG, було висвітлено процес побудови віртуального прототипу. Також був представлений найскладніший етап розбиття дизайну, оскільки запропоновано новий алгоритм обробки величезної кількості транзакцій, отриманих на етапі моделювання. Основні транзакційні потоки були визначені та застосовані до прототипу декодера з

метою відображення основних шляхів даних у дизайні. Поділ транзакцій на потоки або посилання робить процес налагодження простішим, даючи можливість швидше знаходити помилки і скоротити час перевірки складних проектів в цілому.

Як результат, місце та ефективність нового методу трансакційного аналізу було показано в процесі перевірки SoC на рівні проектування системи.

ВИСНОВКИ

Основна мета роботи була виконана в ході теоретичних та експериментальних досліджень. Найголовніша проблема була вирішена - пошук нових ефективних методів функціональної перевірки на рівні проектування системи. Знайдене рішення у вигляді розробленого алгоритму майнінгу частотного шаблону для бази даних TLM показало свою ефективність у аналізі зв’язків реальних проектів.

Для цього було створено експериментальне комплексне рішення для верифікації. Він складається з оригінального ядра моделювання, що дозволяє взаємозв’язувати компоненти SoC на рівні транзакцій. Як середовище даних транзакцій, що передають протокол мережевого зв'язку [35].

Рішення для перевірки Aldec® Riviera ™ було ввімкнено для використання моделей транзакцій у процесі моделювання. Будучи інтегрованим у розроблене середовище верифікації, цей симулятор виявив можливість підключення будь-якого інструменту EDA зі стандартною підтримкою SystemC до запропонованого рішення IVE.

В результаті проект IVE може використовуватися не замість існуючих рішень, а на додаток до них, як просте середовище моделювання та верифікації на рівні системи. Команди розширеної верифікації повинні використовувати не одне рішення для перевірки та моделювання складних проектів. Кожен етап проектування потребує свого випробування. Ось чому рішення IVE може бути використане на ранніх стадіях проектування потоків SoC, таких як віртуальне прототипування та створення транзакційних моделей.

Для того, щоб випустити комерційну версію продукту, яка була б привабливою при розгортанні підходу до проектування та перевірки на системному рівні, слід виконати наступні дії:

- удосконалення алгоритму TLM-FP для кращої візуалізації

транзакційного аналізу;

- надавати більше можливостей аналізу транзакцій у вигляді опцій плагіна;

- додавати локальний модельний компроміс за допомогою каналів пам'яті, що дозволяє прискорити процес моделювання на одній машині, коли моделювання мережі не є необхідним;

- реалізувати редактор блок-діаграм (BDE), щоб надати можливість збирати прототипи SoC у графічному, більш очевидному вигляді;

- забезпечити IP-бібліотеку відповідними моделями транзакцій SystemC, що є звичним для постачальників EDA.

У цьому контексті рішення для верифікації слід сприймати як сукупність утиліт, які допомагають досягти цілей верифікації. І процес верифікації повинен базуватися на методології, щоб зібрати всі необхідні інструменти для досягнення мети проекту.

Підводячи підсумок, протягом всього процесу розслідування були досягнуті наступні цілі:

- дослідження проблематики за допомогою вивчення численної спеціалізованої технічної літератури;

- дослідження методів взаємозв'язку для блоків в SoC, описаних на різному рівні абстракції;

- було описано нове бачення процесу моделювання системи та методів передачі даних у запропонованому середовищі моделювання;

- на основі демонстраційного проекту було показано гнучке прототипування віртуальної системи;

- оригінальна архітектурна система моделювання була побудована на основі верифікаційного рішення Aldec® Riviera ™;

- розроблений формат інтерфейсів XML IP-адрес;

- винайдено новий алгоритм аналізу зв'язку транзакційних даних;

- проведено оцінку процесу верифікації проекту JPEG Decoder;

- майбутні плани подальших розслідувань були введені з метою

реалізації запропонованого рішення комерційним шляхом.

Вважається, що, незважаючи на експериментальний характер, пропоноване рішення IVE із методом аналізу транзакційних даних може бути використане повністю або частково як додатковий ресурс верифікації на рівні проектування системи спільно зі звичайним симулятором RTL для досягнення кращої якості продукту за короткий час.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. L. Bening, H. Foster, Principles of Verifiable RTL Design, Kluwer Academic Publishers, 2000.

2. Smith, D. J. HDL Chip Design: A Practical Guide for Designing, Synthesizing, and Simulating ASICs and FPGAs using VHDL or Verilog, Madison, 448 p., 1996.

3. Open SystemC Initiative, SystemC Homepage, www.systemc.org.

4. Mentor Graphics European ESL Survey 2005, www.mentor.com.

5. Wilcox P., Professional Verification. A Guide to Advanced Functional Verification, Cadence Design Systems, Inc., pp. 41-69, 2004.

6. Reis R., Jochan A. G. Jess. Design of System on a Chip: Devices And Components, Chapter 1, Kluwer Academic Publishers, pp. 8-12, 2004.

7. A Vijay, K. Madisetti, Chonlameth Arpikanondt, Platform-Centric Approach to System-on-Chip (SOC) Design, Chapter 1, Springer, pp. 17-21, 2005.

8. J. Bhasker. A SystemC Primer, Star Galaxy Publishing, Chapter 1, pp. 1-13, June 2002.

9. Rashinkar P., Paterson P., Singh L. System-on-a-Chip Verification - Methodology and Techniques, Kluwer Academic Publishers, pp. 7-35, 2002.

10. Andreas S. Meyer Principles of functional verification, Elsevier Science, 206 p., 2004.

11. HES Acceleration Datasheet, http://www.aldec.com/products/hes.

12. Ghenassia F. Transaction Level Modeling with SystemC. TLM: An Overview and Brief History, Springer, pp. 1-14, 2005.

13. B. C. Cole. Getting to the Market On Time, Electronics, pp. 62-65, April 1989.

14. Bailey B., Martin G. and Piziali A. ESL Design and Verification: A Prescription for Electronic System Level Methodology, Morgan Kaufmann/Elsevier, 2007.

15. Lennard C., Mista D. Taking Design to the System Level, ARM White Paper, April 2005.

16. Draft Standard SystemC Language Reference Manual, 25 April 2005

17. SpecC Technology Open consortium, SpecC – Objective,

18. Goering R. Tools ease transaction-level modeling, EETimes, January 2006.

19. Ghenassia F. Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems, Springer, pp. 14-67, 2005.

20. Martin G. SystemC Tools, Cadence Berkley Labs, October 2002.

21. Haverinen A., Leclercq M., Weyrich N., Wingard D. SystemC-based SoC Communication Modeling for the OCP Protocol, 2002, (http://www.ocpip.org/socket/whitepapers).

22. Lin T. Y., Ohsuga S., Liau C. J., Hu X., Tsumoto S. Foundations of Data Mining and Knowledge Discovery, Springer-Verlag Berlin Heidelberg, 2005.

23. Borgelt C., An Implementation of the FP-growth Algorithm, OSDM'05, 2005.

24. Mao R., Adaptive-FP: An Efficient and Effective Method for Milti-Level Multi-Dimensional Frequent Pattern Mining, Simon Fraser University, April 2001.

25. Stuijk S. Design and Implementation of a JPEG Decoder, Eindhoven University of Technology, December 2001.

26. Free open source IP-cores and chip designs: http://www.opencores.org, 2006.