Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти____другий (магістерський)_____
(рівень вищої освіти)

Моделі та методи верифікації автоматних систем логічного управління
(тема)

Виконав: студент 2 курсу, групи СКСм-18-2

_____Фоменко В.В._____
(прізвище, ініціали)

Спеціальність *123 _ Комп'ютерна інженерія*

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

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи _____доц. Шкіль О.С.___
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_ Комп'ютерна інженерія _____
(шифр і назва)

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

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
_____ (підпис)
«_____»_____ 20 ___р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові_____Фоменко Владиславу Володимировичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Моделі та методи верифікації автоматних систем логічного управління_____
_____
Testable Logical Control Systems Verification Models and Methods Based on Finite State Machines_____

затверджена наказом по університету від "_04_"__11__2018 р. № _1624 Ст_____.___

2. Термін подання студентом роботи до екзаменаційної комісії _____20.12.2019_____

3. Вихідні дані до роботи (проекту)_____
Мова опису апаратури VHDL_____
Система моделювання Active-HDL_____
САПР XILINX ISE_____
ПЛІС Spartan 3E_____
_____
_____

4. Перелік питань, що потрібно опрацювати у роботі _____
Методи проектування систем логічногго управління_____
Методи автоматного програмування при побудові моделейі автоматів_____
Класифікація методів верифікації проектів систем логічного управління _____
Моделювання та синтез HDL-моделей кінцевих автоматів_____
Верифікація HDL-моделей кінцевих автоматів у реальному часі_____
_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 23 слайди

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання 03.09.2018

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Видача теми проекту, узгодження і затвердження	03.09.2019 -10.09.2019	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	10.09.2019 -30.09.2019	
3	Проектування систем логічного управління	30.09.2019 -15.10.2019	
4	Аналіз методів верифікації проектів систем логічного управління	15.10.2019 -15.11.2019	
5	Розробка процедур перевірки графових автоматних моделей на коректність	15.11.2019 - 25.11.2019	
6	Побудова та верифікація HDL-моделей пртистроя керування	25.11.2019 -05.12.2019	
7	Розробка системи апаратного налагодження проектів на ПЛІС	05.12.2019 -15.12.2019	
8	Оформлення пояснювальної записки	15.12.2019 -20.12.2019	
9	Захист проекту	20.12.2019 -25.12.2019	

Студент _____
(підпис)

Керівник роботи (проекту) _____ доц. Шкіль О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 84 сторінки, 43 рисунки, 2 таблиці та 13 джерел за переліком посилань.

КІНЦЕВИЙ АВТОМАТ, СТАН, ГРАФ ПЕРЕХОДІВ, САПР, МОВА ОПИСУ АПАРАТУРИ.

Метою роботи є розробка моделей та процедур верифікації в циклі автоматизованого проектування автоматних систем логічного управління на технологічній платформі ПЛІС з використанням мов опису апаратури...

Проведений аналіз існуючих методик проведення верифікації моделей автоматних пристроїв логічного керування на різних стадіях автоматизованого проектування. Розроблені процедури перевірки графових моделей керуючих автоматів на коректність на алгоритмічному етапі автоматизованого проектування, а також процедури апаратної верифікації проектів керуючих автоматів на технологічній платформі ПЛІС. Розроблені методики верифікації реалізовані в рамках САПР на основі мов опису апаратури для пристрою логічного керування.

ABSTRACT

The explanatory note contains: 84 pages, 43 figures, 13 sources according to the list of links.

FINITE STATE MACHINE, STATE, STATE DIAGRAM, CAD, , HARDWARE DESCRIPTION LANGUAGE.

The purpose of the work is to develop models and procedures for verification in the cycle of computer-aided design of automated logic control systems on the FPGA technology platform using the hardware description language..

The analysis of existing methods of verification of models of finite state machines devices of logical control at different stages of computer aided design is carried out. Procedures for verification of graph models of control finite state machines for correctness at the algorithmic stage of computer aided design, as well as procedures for hardware verification of control finite state machine projects on the FPGA technology platform have been developed. The developed verification methods are implemented within the CAD based on the hardware description language.. for the logic control device.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.……7
ВСТУП……..……..8
1 МЕТОДИ ТА ПРОЦЕДУРИ ПРОЕКТУВАННЯ СИСТЕМ ЛОГІЧНОГО УПРАВЛІННЯ…..10
1.1 Особливості побудови систем логічного управління.....10
1.1.1 Автоматні системи логічного управління……10
1.1.2 Методика побудови графових моделей керуючих автоматів……14
1.2 Часові автомати в системах управління реального часу ……19
1.3 Верифікація в циклі проектування керуючих автоматів……..…26
2 ТЕХНОЛОГІЇ ВЕРИФІКАЦІЇ МОДЕЛЕЙ ЧАСОВИХ АВТОМАТІВ НА ПЛІС……35
2.1 Перевірка графових моделей керуючих автоматів на коректність…...35
2.1.1 Ортогоналізація логічних функцій умов переходів……....36
2.1.2 Аналіз коректності HDL-моделей керуючих автоматів при автоматизованому синтезі……..42
2.2 Технології функціональної верифікації в САПР XLINX ISE…….....49
2.3 Апаратна верифікація проектів керуючих автоматів в ПЛІС ……..…57
3 РЕАЛІЗАЦІЯ МЕТОДІВ ВЕРИФІКАЦІЇ В СИСТЕМАХ ЛОГІЧНОГО УПРАВЛІННЯ НА ПЛІС…….…….64
3.1 Графові моделі автоматних пристроїв керування та їх верифікація …64
3.2 Функціональна верифікація HDL-моделей автоматних пристроїв керування ……..….69
3.3 Апаратне тестування пристрою логічного керування на ПЛІС у реальному часі……..……72
ВИСНОВКИ……..……....81
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ…….……..…83
ДОДАТОК А Графічна частина атестаційної роботи……....85
ДОДАТОК Б HDL-модель пристрою керування тепловентилятором…….97
ДОДАТОК В Програмний TestBench для пристрою керування тепловентилятором……..……99
ДОДАТОК Г Пристрій апаратної верифікації для системи керування тепло вентилятором…….……..102

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГСА – граф-схеми алгоритмів;

ДП – дискретний пристрій;

ДНФ – диз'юнктивна нормальна форма

КА – керуючий автомат;

ОА – операційний автомат;

СЛУ – системи логічного управління

ПЗ – програмне забезпечення;

ПЛІС – програмовані логічні інтегральні схеми;

РЕП – радіоелектронні пристрої;

САПР – система автоматизованого проектування;

ЦП – цифровий пристрій;

CPLD – Complex Programmable Logic Device (різновид ПЛІС);

FPGA – field-programmable gate array (різновид ПЛІС);

FSM – finite state machine (кінцевий автомат);

HDL – hardware description language (мова опису апаратури);

RTL – register transfer level (рівень регістрових передач);

SoC – System of Chip (система на кристалі);

TDI – test data input (вхід подачі тестових сигналів);

TLM – Transaction-level model (моделі рівня трансакцій);

UUT – Unit Under Test (об'єкт діагностування);

UVM – Universal Verification Methodology (універсальна методологія верифікації);

VHDL – very high speed integrated circuits HDL (одна з мов опису апаратури).

ВСТУП

Основними проблемами розвитку ринку сучасної мікроелектроніки є зниження вартості і скорочення часу проектування, що досягається вдосконаленням технічного, інформаційного і програмного забезпечення систем автоматизованого проектування радіоелектронної апаратури (САПР РЕА). Найбільш складним і витратним етапом в сучасному циклі проектування цифрових пристроїв (ЦП) є функціональна верифікація, тобто процес виявлення, локалізації та усунення помилок в моделях різного рівня абстракції специфікації, на що витрачається до 70% загального часу проектування. Основною формою опису проектів цифрових пристроїв в САПР РЕА є мови опису апаратури (Hardware Description Language, HDL), тому об'єктом верифікації є модель цифрового пристрою, яка написана на мові опису апаратури, тобто HDL-код. Особливої ваги набуває верифікація HDL-моделей на ранніх стадіях проектування, що дозволяє скоротити час на повний цикл проектування, і знизити витрати на автоматизоване проектування в цілому. Важливою складовою частиною процесу верифікації HDL-моделей є пошук і виправлення помилок проектування в разі відмінності між результатами моделювання HDL-коду та специфікацією на проектований пристрій.

В останні роки верифікація проектів цифрових систем, заданих на мовах опису апаратури VHDL або Verilog, стає найбільш актуальним завданням у всьому циклі проектування. Інженер-розробник має бути впевнений, що створений ним опис не містить алгоритмічних або будь-яких інших несінтаксіческіх помилок, не охоплених компілятором. Крім цього такі описи в результаті синтезуються в цифрові системи на сучасній елементній базі, налагодження яких вимагає спеціального обладнання і матеріальних витрат. З іншого боку, перевіривши опис на етапі, що передує «прошивці» різних типів програмованих логічних інтегральних схем (ПЛІС): Field

Programmable Gate Array (FPGA) або Complex Programmable Logic Device (CPLD), розробник може бути практично впевнений, що створюваний пристрій запрацює так, як він припускав. Очевидно, що при виявленні помилки простіше виправити кілька рядків коду, ніж заново «перепрошивати» весь кристал при виявленні невідповідності опису і технічного завдання (ТЗ).

Таким чином, розробка методів верифікації моделей цифрових пристроїв на різних стадіях циклу проектування та різних рівнях абстракції є актуальною та технічно значущою задачею.

Об’єкт дослідження у роботі – процедури автоматизованого проектування пристроїв логічного керування на технологічній платформі ПЛІС.

Предмет дослідження – моделі, методи та процедури верифікації автоматних моделей пристроїв логічного керування, реалізованих в ПЛІС, на різних етапах автоматизованого проектування.

Мета роботи – розробка моделей та процедур верифікації в циклі автоматизованого проектування автоматних систем логічного управління на технологічній платформі ПЛІС з використанням мов опису апаратури.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз існуючих методик проведення верифікації ена різни хетапах автоматизованого проектування цифрових систем;

– провести подальший розвиток процедур проектування автоматних систем логічного керування з використанням мов опису апаратури;

– розробити процедури перевірки графових моделей керуючих автоматів (КА) на коректність на алгоритмічному етапі автоматизованого проектування

– розробити процедури апаратної верифікації проектів керуючих автоматів на технологічній платформі ПЛІС;

– реалізувати розроблені методики верифікації в рамках САПР на основі мов опису апаратури.

1 МЕТОДИ ТА ПРОЦЕДУРИ ПРОЕКТУВАННЯ СИСТЕМ ЛОГІЧНОГО УПРАВЛІННЯ

1.1 Особливості побудови систем логічного управління

1.1.1 Автоматні системи логічного управління

Одним з класів систем автоматичного управління є системи логічного управління (СЛУ). Специфіка СЛУ полягає в тому, що у них вхідні сигнали X і вихідні сигнали Y можуть приймати тільки два значення - 0 та 1.

Об'єктами управління (ОУ) для систем логічного управління є клапани, вентилятори, електродвигуни, пристрої освітлення, пристрої регулювання температури, вологості, тиску тощо.

СЛУ впливають на ОУ не безпосередньо, а через виконавчі механізми – магнітні пускачі, електромеханічні перетворювачі, гідроромеханічні перетворювачі, регулятори тиску, терморегулятори тощо. Тому при побудові таких систем передбачається, що вони входять до складу ОУ.

Інформація про роботу ОУ надходить до СЛУ від сигналізаторів положення, стану і параметрів (датчиків та сенсорів). При цьому необхідно зазначити, що сигналізатори параметрів складаються з двох складових - датчика і вторинного приладу (перетворювача), який здійснює порівняння значення аналогового сигналу від датчика із заданими значеннями параметрів, і формування двійкового сигналу {0, 1}. При цьому «0» визначається виходом значення за межі встановлених діапазонів, а «1» знаходженням значення параметру у заданому діапазоні [1].

Загальна структура системи автоматичного управління показана на рис.1.

В основі моделей систем логічного управління лежить поняття «стану». Стан це математична абстракція, яка однозначно ставиться у відповідність

кожному з фізичних станів об'єкта управляння, так як зазвичай функціонування технічних систем проявляється через зміну їх станів. При цьому кожен стан в алгоритмі управління підтримує об'єкт управління в належному стані, а перехід в новий стан в алгоритмі призводить до переходу об'єкта в новий відповідний стан, що і забезпечує процес логічного управління об'єктом [2].

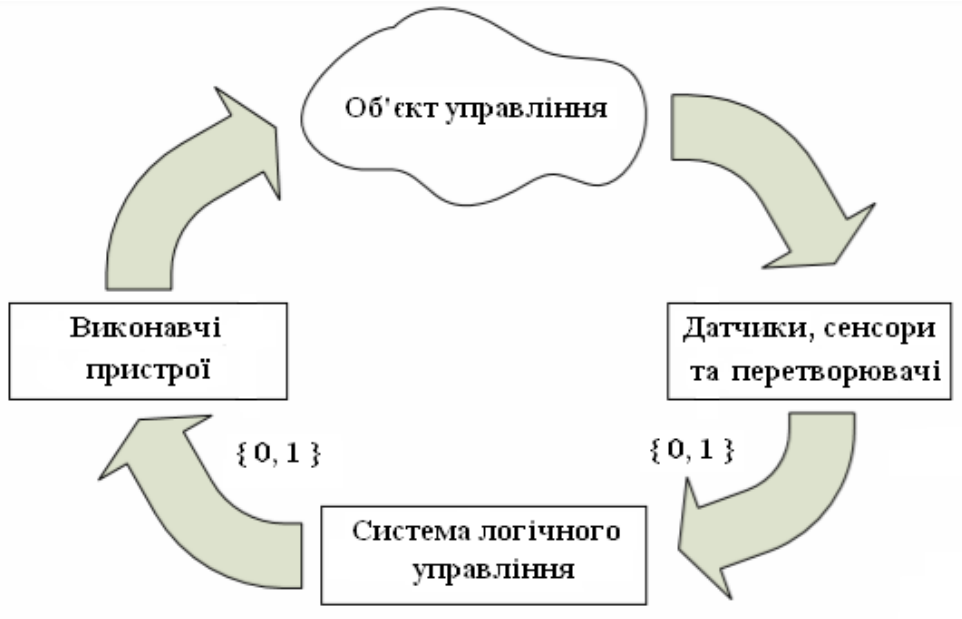

Рисунок 1.1 – Загальна структура системи автоматичного управління

Стан – це сукупність параметрів технічної системи в даний момент часу. Поточний стан несе в собі усю інформацію про минуле системи, необхідну для визначення її реакції на будь-яку вхідну дію, що формується у даний момент часу. Стан можна розглядати як особливу характеристику, яка в неявній формі об'єднує усі вхідні дії минулого, що впливають на реакцію системи. Реакція системи у цій термінології залежить тільки від вхідної дії і поточного стану. Таким чином стан можна визначити як сукупність параметрів моделі динамічної технічної системи, значення яких залежать від вхідних сигналів та передісторії даних параметрів. Кожен стан технічної системи, окрім сукупності своїх параметрів, характеризується часом

знаходження системи у зазначеному стані (Т). Перейти у новий стан система може по закінченню часу Т (при наявності відповідних значень вхідних сигналів) або за подією, яка безпосередньо змінює стан технічної системи.

Основна властивість стану системи в даний момент часу t_0 полягає в «відділенні» майбутнього ($t > t_0$) від минулого ($t < t_0$) в тому сенсі, що стан несе в собі всю інформацію про минуле системи, яка необхідна для визначення реакції системи на будь-який вхідний сигнал, що подається в момент t_0.

В основі функціонування моделей систем логічного управління лежить поняття «кінцевий автомат». Кінцевим автоматом називається математична модель синхронної системи для перетворення вхідної інформації, представленої кінцевим алфавітом у вихідну інформацію, представлену також кінцевим алфавітом через кінцеве число стійких станів [3]. Абстрактний кінцевий автомат представляється шестіркою $W =< X, A, Y, \delta, \lambda, a_0 >$, де $X = \{x_1, x_2, ..., x_m\}$ – множина букв вхідного алфавіту; $A = \{a_1, a_2, ..., a_n\}$ – множина станів автомату; $Y = \{y_1, y_2, ..., y_r\}$ – множина букв вихідного алфавіту; $\delta(a_i, x_k) = a_j$ – функція переходів автомату, $\lambda(a_i, x_k) = y_\alpha$ – функція виходів автомату, a_0 – початковий стан автомату.

Структурні моделі автоматів, які використовуються в задачах логічного управління, можуть розглядатися як пристрої керування. Тому в усіх структурних моделях існує множина вхідних сигналів X, множина станів A і множина вихідних реакцій Y (усі три множини є кінцевими). У завданнях логічного управління значення має не лише кінцева множина X, A і Y, але і їх точна розмірність, оскільки вона впливає на складність реалізації пристрою керування. Символам вхідного та вихідного алфавітів ставляться у відповідність вектори вхідних (X) та вихідних (Y) змінних, а при схемній реалізації – зовнішніх входів та виходів схеми. Кожен стан абстрактного

автомата a_i кодується вектором внутрішніх змінних $Z_i = \{z_{i,1}, z_{i,2},..., z_{i,k}\}$.

Поняття «значення вхідних сигналів» (input values, входные воздействия), поділяється на вхідні дії («input actions», «входные воздействия») та події (events, события). Вхідні дії реалізуються автоматом шляхом опитування у відповідності до алгоритму його роботи в циклі управління СЛУ, а події (або набір подій), реалізуються миттєво та призводять до зміни стану автомата.

Поняття «значення вихідних сигналів» або вихідні реакції (output values, выходные реакции), в свою чергу поділяється на «вихідні дії» («output actions», «выходные действия») і «вихідну діяльність» («output activity», «выходную деятельность»), Передбачається, що дія є одноразовою, миттєвою і неперервною, а діяльність може тривати досить довго і можливо її переривання деякою вхідною подією [2].

Вхідний сигнал в загальному випадку може змінити стан; ініціювати вихідний сигнал без зміни стану; ініціювати вихідний сигнал і змінити стан. При цьому слід вважати, що реакцією на подію може бути тільки один перехід.

При побудові систем автоматичного управління як правило, виділяють пристрої, що керують, і керовані об'єкти. Виходячи з цього системи логічного управління можна розділити на дві частини:

– частину, що керує, відповідальну за логіку поведінки, тобто за вибір виконуваних дій, залежний від поточного стану і вхідних сигналів, а також за перехід в новий стан;

– керовану частину, відповідальну за виконання дій, вибраних для виконання керуючою частиною, і, можливо, за формування деяких компонентів вхідних оповіщувальних сигналів для частини, що керує, тобто зворотних зв'язків.

Відповідно до традиційної теорії автоматичного управління, керована частина визначається як об'єкт управління, а частина, що керує як система управління. Оскільки для реалізації системи управління використовуються

автомати, то вона часто називається керуючий автомат або просто автомат. Сукупність об’єкту управління та керуючого автомату прийнято визначати як автоматну систему управління.

1.1.2 Методика побудови графових моделей керуючих автоматів

Для системи логічного управління, що розглядається, будується схема інформаційних зв'язків «джерела інформації – керуючий автомат – засоби представлення інформації – виконавчі механізми», що в загальному випадку є схемою із зворотними зв'язками, як це прийнято в теорії автоматичного управління (рис. 1.1).

Кожний інформаційний зв'язок у схемі позначається змінною. При цьому вважається, що вхідні змінні керуючого автомата належать множині двійкових змінних X, а його вихідні змінні - множині двійкових змінних Y. Схема може містити також короткі коментарі, що пояснюють сенс використовуваних змінних і властивості органів управління, виконавчих механізмів і власне об'єкта управління.

2. Вводиться поняття «стан» об'єкта управління. Визначаються і перераховуються (класифікуються) його стани, що формує простір станів об'єкта. Стани об'єкта управління можуть бути як стійкими, так і нестійкими. Так як зазвичай об'єкти управління при автоматизації технологічних процесів є дуже інерційними в порівнянні з часом програмного циклу керуючого обчислювального пристрою, то тому вони не тільки в стійких, але і в нестійких станах можуть перебувати досить «довго». Наприклад, для такого об'єкта, як клапан, стійкими є такі його стану, як «закритий» і «відкритий», а нестійкими - «відкривається» і «закривається». Відзначимо, що при перемиканні клапана його сигналізатори не дозволяють відрізнити послідовність стійких станів «закритий» - в «відкритий» від послідовності «відкритий» - «закритий». Нестійкі стану клапана «відкривається» і «закривається» дозволяють розрізнити ці послідовності.

Чотири стану можна виділити також і в такому об'єкті управління, як лампа, два з яких стійкі («не горить» і «горить»), а два - нестійкі («загоряється» і «гаситься»). При цьому відзначимо, що якщо для клапана стійкі стану можуть бути відокремлені від нестійких за допомогою сигналізаторів, то для лампи в кожній парі станів «горить» - загоряється »і« не горить »- гаситься» в стійкий стан від нестійкого по зовнішньому прояву невиразні. Нестійкі стани в цьому випадку введені для того, щоб відрізнити послідовність стійких станів «не горить» -> «горить» від послідовності «горить» -> «не горить». Нестійким станам лампи в подальшому будуть відповідати стійкі стану керуючого нею автомата, які зберігаються при одиничному (нестійкому) значенні вхідної змінної, а стійким станам лампи - стійкі стану автомата, які зберігаються при нульовому (стійкому) значенні тієї ж вхідний змінної. Якщо послідовність зміни стійких станів, наприклад, лампи несуттєва, то її нестійкі стани можуть не розглядатися, а керуючий автомат стає комбінаційною схемою.

У разі, коли потрібно врахувати можливі несправності об'єкта, простір його станів повинно бути розширено. При цьому якщо вибір числа «нормальних» станів є в деякому сенсі об'єктивним, то вибір числа додаткових станів суб'єктивний і залежить від бажання розробника алгоритму. Зі збільшенням числа додаткових станів об'єкта «роздільна» здатність створюваного алгоритму управління збільшується. При цьому, чим більше нюансів розробник хоче відобразити у графі об'єкта, а в подальшому і в алгоритмі управління, тим більше станів повинно бути виділено. Припускаючи, наприклад, що несправність клапана може складатися або в його невідкриття, або незакриття, в граф переходів об'єкта може бути введено лише один додатковий стан «несправність». Якщо ж прийнято рішення відрізнити невідкриття від незакриття, то замість одного стану повинні використовуватися два стани, кожен з яких відповідає «своїй» несправності: «невідкриття» і «незакриття».Таким чином, множина станів об'єкта може складатися з двох підмножин: «норма» і «несправності».

Кожному стану об'єкта зіставляється вершина в графі об'єкта. Вершини розташовуються на площині і для ідентифікації нумеруються (іменуються) числами (іменами) від 0 до (s-1), де s - число виділених станів об'єкта.

Кожна вершина графа об'єкта помічається підмножиною значень двійкових змінних, з множини вхідних змінних X, які формуються сигналізаторами об'єкта в цьому стані. Позначка вершин з підмножини «норма» визначається властивостями об'єкта, а вершин з підмножини «несправності» - рішенням розробника про те, яка інформація характеризує відповідний стан. При цьому різні вершини графа об'єкта можуть бути позначені однаковими змінними.

Визначаються всі допустимі переходи між станами об'єкта, що відбивається введенням відповідних дуг в граф об'єкта. Дуги між вершинами з підмножини «норма» вводяться відповідно до властивостей об'єкта, а дуги між вершинами з підмножин «норма» і «несправності» вводяться розробником в залежності від прийнятих на другому і четвертому етапах методики вирішенні про ідентифікацію несправностей і рішенні про те, що робити після їх усунення. При цьому для кожної вершини графа, що відповідає стійкому стану об'єкта, вводиться петля.

6. Кожна дуга і петля в графі об'єкта позначається кон'юнкцією змінних або їх інверсій з підмножини множини вихідних змінних Y, які відповідають значенням змінних, що подаються на входи виконавчих механізмів об'єкта і засобів представлення інформації. У цьому графі у нестійких вершинах вихідні дуги можуть бути помічені однаково. На цьому завершується побудова графа об'єкта.

На основі графу об'єкта будується модель об'єкта, яку будемо називати графом переходів функціонування моделі (граф моделі). У графі моделі повністю зберігається «скелет» графа об'єкта та позначки його вершин. У графі моделі все вершини повинні бути стійкими, і тому кожна його вершина повинна мати інцидентну їй петлю.

У кожну вершину до значень змінних з множини X додаються

значення двійкових змінних з множини T (часових змінних), які моделюють час перебування об'єкта у відповідних станах. Кількість цих змінних може бути зведена до однієї, якщо можна зробити припущення про те, що всі перехідні процеси в об'єкті мають однакову тривалість.

Дуги графа моделі, які виходять із вершин, що відповідають стійким вершин графа об'єкта, позначаються прямими або інверсними змінними з множини Z, що входять до складу кон'юнкцій, які ініціюють відповідні переходи в графі об'єкта.

Кожна дуга графа моделі, яка виходить із вершини, відповідної нестійкою вершині графа об'єкта, позначається кон'юнкція, що складається з двійковій змінної з множини T, яка визначає факт спрацьовування функціонального елемента затримки, і з мінімального числа довічних змінних або їх інверсій з підмножини безлічі X, що характеризують стан об'єкта в суміжній стійкою вершині.

При необхідності здійснюється позначка петель вершин графа моделі за рахунок забезпечення повноти переходів з кожної вершини.

8. Побудова графа переходів автомата починається з аналізу технічного завдання з метою виявлення необхідності використання часових параметрів в алгоритмі управління. Якщо така необхідність відсутня, то будується нова схема зв'язків, що відрізняється від вихідної заміною словосполучення «керуючий автомат» на слово «автомат».

У разі, коли в завданні згадуються часові затримки, будується нова схема зв'язків, що відрізняється від вихідної тим, що в ній керуючий автомат декомпозирован на автомат і часові змінні, число яких дорівнює числу різних часових затримок, що згадуються в завданні. При цьому для автомата вводяться два множини двійкових змінних t і T, описаних в попередньому пункті, перше з яких для автомата є вихідним, а друге - вхідним.

Проведення мінімізації числа станів автоматів буває необхідно тільки при жорстких обмеженнях на обсяг пам'яті і швидкодії і недоцільно за відсутності таких обмежень. Більш того, при мінімізації числа станів

структура графа автомата починає «істотно» відрізнятися від структури графа об'єкта, що, наприклад, вкрай незручно, якщо потрібно виконувати сигналізацію станів об'єкта не від його сигналізаторів, а використовуючи побудований автомат. Відзначимо, що мінімізація числа станів автомата може проводитися не тільки поєднанням вершин його графа переходів, а й, наприклад, за рахунок побудови на його основі головного і викличного графів, останній з яких реалізує однотипні фрагменти заданого графа.

Вище розглядалися стани автомата, які прямо (для автоматів без мінімізації числа станів) або узагальнено (для автоматів з мінімізованих числом станів) були пов'язані з станами справного або несправного об'єкта управління. Однак автомати можуть містити також і такі стани, які з станами керованого об'єкта не пов'язані, а їх введення може визначатися, наприклад, необхідністю відображення в графі переходів неправильних дій оператора. Облік таких станів вимагає коригування схеми зв'язків за рахунок введення в загальному випадку додаткових органів управління, засобів представлення інформації і функціональних елементів затримки.

Побудована схема зв'язків і граф переходів автомата, що входить в цю схему, у вичерпній, однозначній і компактній формі визначають технічне завдання на розробку керуючого пристрою. При цьому словесний опис може використовуватися лише для напису коментарів, так як повністю навіть простий граф переходів словами описати досить складно і громіздко.

Викладена методика відповідає на питання: звідки «беруться» стани в керуючому автоматі. З її розгляду випливає, що якщо поняття «стан» не застосовувати, то при алгоритмізації виникають дуже серйозні труднощі, як це має місце, наприклад, при побудові граф-схем для алгоритмів з пам'яттю, які будуються зазвичай не в результаті аналізу станів автомата, а за рахунок перевірки значень його окремих змінних. Граф-схеми алгоритмів використовуються, як правило, при побудові мікропрограмних керуючих автоматів.

На основі вищерозглянутого відзначимо, що запропонована методика

дозволяє будувати графи переходів для автоматів Мура, які найбільш доцільно використовувати при логічному управлінні такими технологічними об'єктами, як наприклад клапани, насоси, вентилятори, що розглядаються як єдиний об'єкт управління.

Таким чином можна сформулювати наступні етапи загального циклу проектування керуючих автоматів в системах логічного управління.

1. Визначення технічних станів об’єкта управління, принципів роботи датчиків (перетворювачів) інформації про стан об’єкта управління, типи виконавчих механізмів на способи керування ними.

2. Визначення переліку вхідних та вихідних сигналів у кожному стані, систему подій, які впливають на функціонування об’єкта управління, часові параметри обробки вхідних сигналів у кожному стані та появи вихідних сигналів (затримки).

3. Визначення типу пристрою керування, обрання моделі керуючого автомату та побудова графа переходів.

4. Опис моделі пристрою керування на мові опису апаратури (мові програмування) та його автоматизоване проектування на обраній технологічній платформі (ПЛІС або мікроконтролер (МК)).

5. Макетування та фізичне моделювання розробленого пристрою керування у реальному часі.

1.2 Часові автомати в системах управління реального часу

Серед усієї множини систем управління значну частину складають системи логічного управління (Logical Control System), у яких керуючі сигнали приймають значення логічного нуля або одиниці в залежності від граничних значень фізичних величин, що визначають дані параметри. Для технічної реалізації зазначених систем найбільш придатною є модель структурного автомата (Finite State Machines), а візуальним поданням алгоритму функціонування є граф переходів (State Diagram). Відмінною

особливістю автоматів логічного управління є наявність серед вхідних сигналів (Input Values) не тільки оповестітельтних сигналів операційного автомата, а й зовнішніх по відношенню до керованої системі подій зовнішнього світу (external events), які для алгоритму управління є переривань.

Кінцеві керуючі автомати (FSM) функціонують в автоматної часу, яке визначається тактами роботи автомата. Але більшість реальних систем логічного управління взаємодіють із зовнішнім світом в метричному часу, тобто є системами реального часу.

Система управління реального часу - система, в якій результуюча дія (діяльність) залежить не тільки від логічних значень простих керуючих дій, а й від часу, протягом якого ці дії проводяться. Головна відмінність завдань в реальному часі від завдань, не залежних від часу, полягає в тому, що в системах реального часу завдання повинні завершитися в межах заданого проміжку часу, який дозволяє коректно завершити процес обробки даних. Для їх реалізації прийнято використовувати модель часового автомата (timed FSM), яка дозволяє враховувати вплив метричного часу на переходи між технічними станами керованої системи.

Будь-яке локальне цифровий пристрій, що реалізує алгоритм обробки інформації або управління, може бути реалізовано двома способами: апаратним або програмно-апаратним. При апаратному способі реалізації заданий алгоритм описується на мові опису апаратури (HDL) і синтезується інструментальними засобами систем автоматизованого проектування (САПР) в ПЛІС (програмовані логічні інтегральні схеми). Перевагою такого підходу є апаратна гнучкість (можливість реалізувати будь-який алгоритм) і досить велику швидкодію.

При описі алгоритму функціонування цифрових пристроїв логічного керування в САПР цифрових пристроїв одним із стилів написання коду є стиль автоматного програмування. У автоматному програмуванні в якості базового використовується поняття «стан» [3]. Стан це математична

абстракція, яка однозначно ставиться у відповідність кожному з фізичних станів об'єкта управління, так як зазвичай функціонування технічних систем проявляється через зміну їх станів. При цьому кожне стан в алгоритмі управління підтримує об'єкт управління в належному стані, а перехід в новий стан в алгоритмі призводить до переходу об'єкта в нове відповідний стан, що і забезпечує процес логічного управління об'єктом. Стан - сукупність параметрів технічної системи в даний момент часу. Поточний стан несе в собі всю інформацію про минуле системи, необхідну для визначення її реакції на будь-який вхідний дію, яке формується в даний момент часу.

При описі поведінки систем управління реального часу необхідно враховувати часові аспекти в їх поведінці. Для цього модель кінцевого автомата розширюється введенням часової змінної, і вводиться поняття часового автомата (timed automata, timed FSM) [4, 5]. Часова змінна (timed variable) постійно збільшує своє значення і "скидається" в 0 при приході вхідного сигналу і перехід автомата в новий стан. Часові змінні вимірюються в автоматних тактах.

Для опису часових аспектів в автоматної моделі використовуються, як правило, три параметра: часові обмеження t_c (timing constraints), (вхідні) таймаути t_{to} (timeouts) і вихідні затримки t_d (output delays), які іноді називаються вихідними таймаутами. Вхідний таймаут визначає максимальний час очікування вхідного впливу (події) для кожного стану автомата. Якщо вхідний символ ні поданий до закінчення часу очікування, то автомат починає опитування вхідних змінних і може перейти в інший стан. Часові обмеження є інтервалами на переходах, які обмежують час, протягом якого перехід може бути виконаний. Вихідні затримки (вихідні таймаут, вихідні затримки) відображають час, який витрачається автоматом на виконання переходу, тобто вихідний сигнал з'являться на виході через інтервал часу, який визначається вихідний затримкою.

У системах логічного управління поняття «значення вхідних сигналів» (input values), ділиться на вхідні дії (input actions), і події (events). Вхідні дії

реалізуються автоматично шляхом опитування відповідно до алгоритму його роботи в циклі управління, а події реалізуються миттєво і призводять до зміни стану автомата.

Обробка подій в системах реального часу, як правило, визначаються, виходячи з динамічних характеристик процесів управління і пов'язаних з ними подій. Подія - це абстрактне поняття, що припускає таку зміну умов зовнішнього середовища, яке породжує певну реакцію системи. Події можуть генеруватися як зовнішнім середовищем, так і всередині самої системи управління її компонентами.

Існують три основні варіанти взаємодії керуючого автомата з зовнішнім середовищем [2].

1. Події використовуються для взаємодії керуючого і операційного автоматів всередині автоматної системи управління. В цьому випадку, якщо події є винятковими (дві події не можуть відбуватися одночасно), обробка подій нічим не відрізняється від обробки значень вхідних змінних автомата.

2. Події поряд з вхідними змінними забезпечують взаємодію автомата з зовнішнім середовищем Це проектне рішення повинно відображати відмінність між подіями і вхідними змінними: автомат обробляє події в момент його появи, в той час як значення вхідних змінних опитуються автоматом за його власною ініціативою.

3. Кожному події зіставляється окремий стан (перехід) в автоматі. Це рішення підходить тільки для реалізації винятковою моделі подій. Крім того, воно відображає активну роль подій, і той факт, що саме виникнення подій ініціює роботу автомата. Це рішення найкращим чином узгоджується з традиційними подійовими системами, де будь-яка функція виходів пов'язана з вмістом подій.

В залежності від мети та особливостей використання моделей часових автоматів існує безліч модифікацій таких моделей, які по різному враховують як спосіб обробки подій, так і спосіб врахування затримок в станах автомата [5, 6]. Виходячи з особливостей функціонування систем логічного управління

запропонована повна модель структурного часового автомату може бути представлена дев'яткою $W = (X, Y, Z, f, g, z_0, T_c, T_{to}, T_d)$, де: $X = \{X_C, X_E\}$ – множина вхідних змінних, X_C – множина оповіщувальних сигналів від об'єкта керування, X_E – множина зовнішніх подій; $Y = \{Y_C, Y_F\}$ – множина вихідних змінних, Y_C – множина реакцій (керуючих сигналів), Y_F – множина діяльностей (вихідних функцій); Z – множина внутрішніх змінних, які визначають кодування станів автомату; f – функція переходів, g – функція виходів; Z_0 – код початкового стану автомата; $T_c = \{t_{c1}, t_{c2} .. t_{cp}\}$ – множина часових змінних для часових обмежень на кожній дузі графу переходів, де p – кількість дуг у графі переходів, $t_{ci} = \{1, k\}$, k – максимальна кількість тактів обмежень на переходах до i-ї вершини графу переходів в режимі опитування, $k = \{1, \infty\}$, ∞ – відповідає виключно подієвій функції переходів; $T_{to} = \{t_{to1}, t_{to2} .. t_{ton}\}$ – множина часових змінних для timeouts (очікуваній) кожного стану автомата, $t_{toi} = \{1, n\}$ – timeout для кожного стану, n – кількість станів автомата; $T_d = \{t_{d1}, t_{d2} .. t_{dm}\}$ – множина затримок для реалізації відповідного вихідного сигналу, де m – кількість вихідних змінних, $t_{dm} = \{1, l\}$, де l – максимальна кількість тактів для реалізації функцій виходу в зазначеному стані автомата. У загальному випадку, часовий автомат може містити усі три часові параметри, але для конкретної задачі можуть використовуватися часові автомати з одним або двома із зазначених параметрів.

Класичну модель часового автомату (timed FSM), яка складається з трьох часових параметрів < t_c, t_{to}, t_d > не можна безпосередньо віднести до традиційної моделі Мура. Функція виходів подібна до автомату Мура, але вихідний сигнал формується після затримки, а не під час переходу автомата у новий стан. Час появи (зміни) вихідних сигналів прив'язаний до робочого фронту синхросигналу. У запропонованій моделі часового автомату логіка

його роботи виглядає наступним чином.

При переході автомата у поточній стан a_i для нього визначається основний часовий параметр $t_{to}(a_i)$ (timeout), тобто час протягом якого автомат має знаходитися у поточному стані, якщо зовнішня подія достроково не переведе автомат у інший стан. t_{to} визначається у автоматних тактах. Після закінчення часу t_{to} автомат реагує на вхідні сигнали (опитує їх) та переходить у наступний стан.

Вихідні сигнали автомата у поточному стані a_i з'являються на виходах автомата у час, який визначається $t_{dj}(a_i)$ (output delays), тобто вихідними затримками для сигналів y_j у стані a_i. Для кожного з вихідних сигналів y_j вихідна затримка визначається у автоматних тактах і може бути різною. При $y_j=0$ модель часового автомату наближається до класичної моделі Мура.

Обробка зовнішніх подій здійснюється наступним чином. Для кожного стану a_i задаються часові обмеження $t_c(a_i)$ (input constraints), тобто проміжок часу, протягом якого автомат, знаходячись у стані a_i, може обробляти вихідні події. Часові обмеження визначаються у автоматних тактах та обчислюється як $t_c = (t_1-t_0)$, де t_0 – початок «вікна» часових обмежень, t_1 – кінець «вікна» часових обмежень. При $t_1 = \infty$ розглядається часовий автомат без вхідних часових обмежень. Якщо зовнішня подія трапилась поза «вікном» часових обмежень автомат на неї не реагує.

Пристрої логічного управління, побудовані на основі кінцевих автоматів, функціонують в автоматної часу, яке вимірюється в автоматних тактах, тобто дискретних відрізках часу, за яке автомат переходить з одного стану в інший. Тривалість автоматного такту в реальних пристроях, як правило, визначається частотою синхросигналу Clk. Для опису часового автомата використовується темпоральний граф переходів. Всі часові параметри темпорального графа переходів реалізується через петлі, умовами для яких є підрахунок числа тактів Clk, що схемно реалізується лічильником count в ПЛІС [6]. На рис 1.2 приведена відповідність умовного позначення затримок у вершинах графа переходів автомата Мура без петель і з петлями.

Затримка T_i (яка може розглядатися як timeout) здійснюється багатократним переходом із стану в цей же стан, при цьому число переходів визначається числом тактів затримки. Значення лічильника count порівнюється з T_i-1, оскільки при переході в стан aj, автомат один такт знаходиться в ньому до перевірки count, і щоб затримка була точно T_i тактів, необхідно ще T_i-1 тактів повторення.

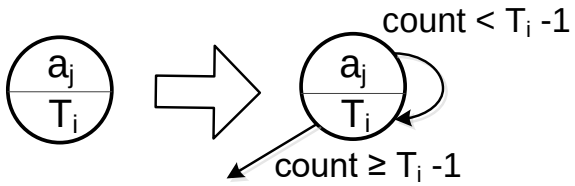

Рисунок 1.2 – Реалізація затримок у темпоральному графі переходів автомата Мура

На рис. 1.3 наведено фрагмент темпорального графа переходів для трьох станів. На рис.1.4 наведено фрагмент часової діаграми функціонування даного автомата.

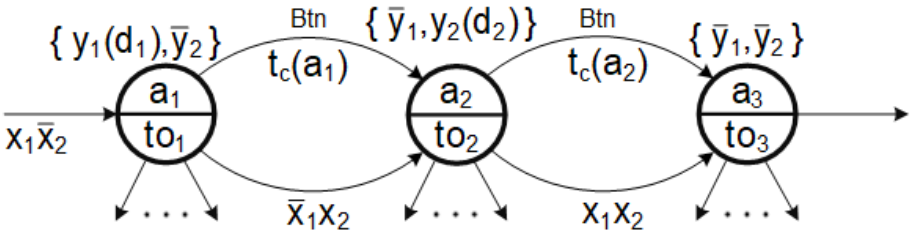

Рисунок 1.3 – Фрагмент в темпорального графа переходів автомата Мура

Для реалізації VHDL-модели часового автомата запропоновано використовувати трехпроцессный шаблон: синхронний процес призначення нового стану, синхронний процес реалізації лічильника автоматних тактів і комбінаційний процес реалізації функції переходів. Функція виходів часового автомата Мура реалізується через оператори умовного призначення вихідних сигналів поза процесами.

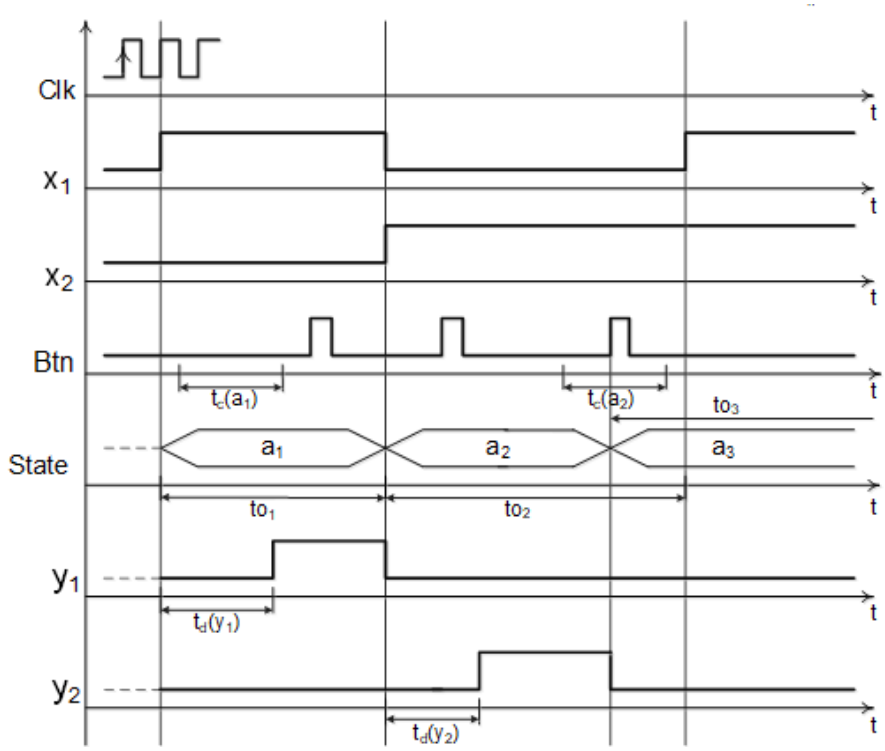

Рисунок 1.4 – Часова діаграма функціонування часового автомата

У загальному випадку, часовий автомат може містити усі три часові параметри, але для конкретної задачі можуть використовуватися часові автомати з одним або двома із зазначених параметрів.

1.3 Верифікація в циклі проектування керуючих автоматів

Верифікація - це процес визначення функціональної коректності моделі при трансформації одного виду опису в інший. Наприклад, щоб в результаті переходу від деякого представлення пристрою у вигляді специфікації до його моделі на мовах опису апаратури не сталося зміни закону функціонування цього пристрою

На жаль, завданню верифікації відводитися мало місця в книгах, присвячених проектуванню на мовах типу VHDL або Verilog. Зазвичай увага приділяється синтаксичним конструкціям, семантиці і стилю, що синтезується, а верифікації присвячена одна глава, да і то верифікацію розглядають як область застосування деяких специфічних мовних

конструкцій. Проте таке відношення до завдання верифікації не відбиває істинний стан справ. У компаніях групи інженерів по верифікації містять майже удвічі більше співробітників, чим групи інженерів - розробників. Крім того, на час проведення верифікації накладаються навіть строгіші обмеження, ніж на час розробки [3]. Зменшити час верифікації можна за рахунок розпаралелювання завдань, використання високих рівнів абстракції, часткової конвеєризації і автоматизації робіт. На жаль, через величезну кількість і різноманітність об'єктів верифікації, методи і підходи цей процес не піддається повній автоматизації. Процес верифікації можна представити моделлю (рис. 1.5), що сходиться.

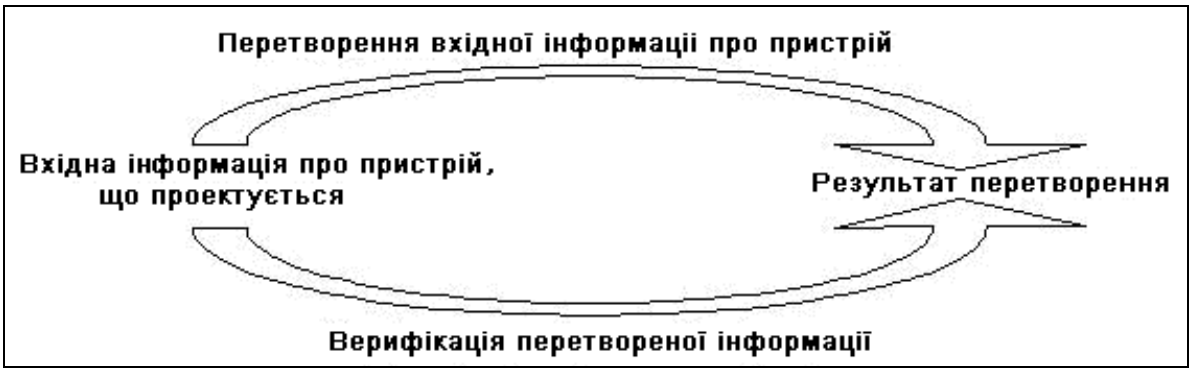

Рисунок 1.5 – Модель процесу верифікації

Процеси перетворення і верифікації повинні мати загальні вхідні і кінцеві точки. Перетворення може виконуватися безпосередньо від внутрішнього опису до VHDL-моделі, або проходити у декілька етапів. Наприклад, якщо спочатку інженер здійснює інтерпретацію специфікації, а потім перетворить своє бачення в код, що синтезується, то при цьому загальною початковою точкою служить результат інтерпретації. Якщо інтерпретація вироблена помилково, то верифікація цього не виявить.

Верифікація може проводитися на програмних емуляторах, на апаратних прискорювачах, або програмно-апаратним способом. Оскільки складність проектів зростає, то слід пам'ятати наступні аспекти верифікації :

– рішення про закінчення проекту не може бути прийняте без проведення його верифікації;

– програмна верифікація має бути проведена до апаратної, оскільки вимагає менше витрат;

– проект повинен створюватися з урахуванням того, що його необхідно буде надалі верифікувати.

У верифікації існують 2 методи: формальна верифікація та тестова верифікація.

Формальна верифікація використовує формальні методи доказу ідентичності моделі до перетворення і потім. Наприклад, чи відповідає список з'єднань, отриманий після етапу синтезу, початковому опису (так звана перевірка еквівалентності). Чи являється перехід від моделі Милі до моделі Мура однозначним. Проте недолік формальної верифікації полягає в низькому рівні здійснення доказу і неможливість застосовувати формальні методи для складних проектів. На жаль, алгоритми формальної верифікації схильні до явища "комбінаторного вибуху" (пов'язане з експоненціальним зростанням простору станів системи), і час рішення на "типових" робочих станціях виявляється неприпустимо великим. Навіть алгоритми, що є лінійними по числу станів, для логічних функцій з великим числом змінних виявляються непридатними.

Другий метод полягає в перевірці того, наскільки отриманий опис відповідає специфікації з точки зору функціонування з використанням заздалегідь підготовлених тестів. Проте оскільки специфікацію пишуть неформальною мовою, то документація може бути інтерпретована по-різному. В цьому випадку тестова верифікація може тільки показати, в якій мірі проект відповідає вимогам специфікації, проте вона не може довести ідентичність проекту і специфікації. Можна сказати, що проект не відповідає специфікації, якщо знайдена хоч одна розбіжність; але не можна сказати, що таких розбіжностей немає.

Основні ідеї і принципи проектування складних систем виражені в

системному підході [7]. Системний підхід включає виявлення структури системи, типізацію зв'язків, визначення атрибутів, аналіз впливу зовнішнього середовища. У функціональному проектуванні переважає низхідний стиль проектування, при якому послідовно виконуються процедури переходу від більш високого рівня абстракції (концепції проекту) до реалізації цифрової системи у вигляді системи на кристалі (System of Chip, SoC). При цьому виділяють такі рівні проектування: системний, алгоритмічний, функціонально-блочний, схемотехнічний, конструкторсько-технологічний.

Для автоматних систем логічного управління на системному рівні аналізуються технічні стани об’єкта управління, визначається перелік вхідних та вихідних сигналів у кожному стані, часові параметри обробки вхідних сигналів у кожному стані та появи вихідних сигналів (затримки).

На алгоритмічному рівні будуються темпоральні графи переходів керуючих автоматів, які є повною моделлю пристрою керування алгоритмічного рівня.

На функціонально-блочному рівні при використанні САПР на технологічній платформі ПЛІС створюється опис алгоритму роботи пристрою керування на мовах опису апаратури (HDL-модель) та його автоматизований синтез.

На схемотехнічному рівні здійснюється імплементація HDL-моделі в кристал ПЛІС а на конструкторському рівні виготовлення фізичного пролторипу пристрою керування.

Процес проектування трансформує множину специфікацій в реалізацію. На рівні опису специфікація визначає функціональність, яка повинна бути реалізована в системі, але не обумовлює способів її реалізації. Можна стверджувати, що як специфікація, так і реалізація є формами опису функціональності. Специфікація визначається на більш високому рівні абстракції, ніж реалізація. В табл. 1.1 наведений перелік етапів низхідного проектування, форми описів на кожному рівні абстракції та процедури верифікації, які застосовуються на кожному з рівнів.

Таблиця 1.1 – Способи верифікації на різних етапах проектування

Рівень проектної абстракції	Форма представлення моделі	Спосіб верифікації
Системний	Специфікація, перелік технічних станів об’єкта	Діалог з Замовником, формування технічного завдання
Алгоритмічний	Граф граф переходів керуючого автомату	Перевірка графа на коректність
Функціонально-блочний	RTL-модель на мовах опису апаратури та результати її синтезу	Функціональне моделювання та постсинтезне моделювання
Схемотехнічний	Результат імплементації схеми в кристал ПЛІС	Моделювання з реальними часовими затримками (timing simulation)
Конструкторсько-технологычний	Прототип пристрою керування	Фізичне тестування прототипу у реальному часі

Якщо у рамках одного рівня наявність декількох ітерацій є очікуваним ефектом, то повернення на рівень назад свідчить про істотні прорахунки проектування і призводить до значних часових і фінансових втрат.

В процесі проектування опис поетапно трансформується від найбільш високого рівня абстракції до найнижчого. Функціональна верифікація - це перевірка відповідності між уточненими описом функціональності на більш низькому рівні абстракції і раніше створеним описом на більш високому рівні абстракції [8] (рис. 1.6):

На ранніх етапах верифікації моделі здійснюється доказ властивостей (property checking), в той час як на більш пізніх етапах ключова роль відводиться перевірці еквівалентності (equivalence checking).

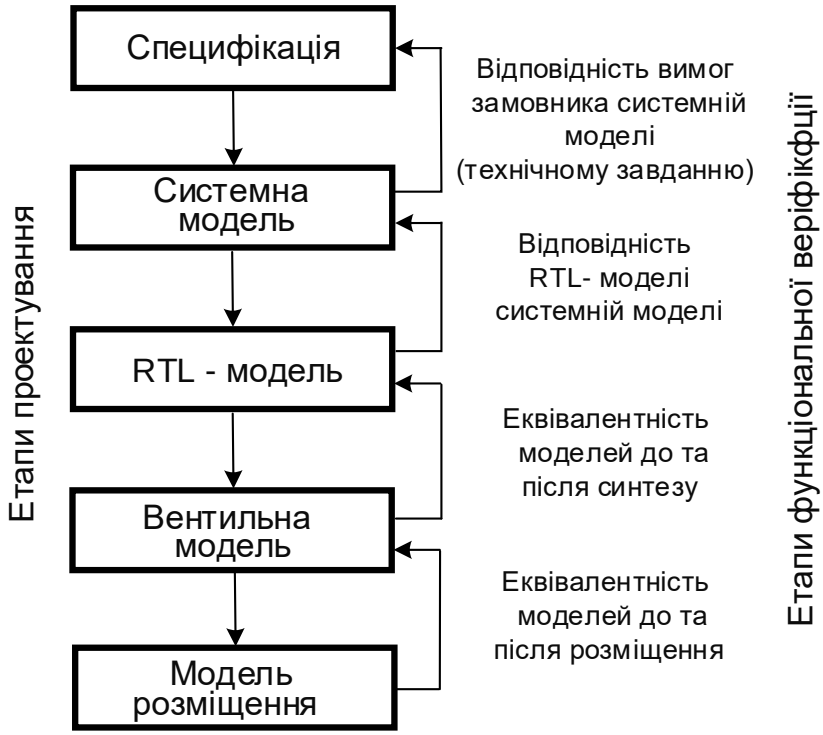

Рисунок 1.6 –Верифікація на різних етапах проектування

Всю множину допустимих функціональних помилок можна поділулити на два основні типи :

– помилки, які характерні для реалізації, але не характерні специфікації - вносяться при реалізації;

– помилки в самій специфікації, які проявляються у вигляді відсутності специфікації опису поведінки в деяких особливих ситуаціях, конфлікт вимог, недокументована поведінка.

Помилки першого типу можуть бути виявлені різними автоматизованими методами функціональної верифікації. Помилки другого роду можуть бути виявлені тільки вручну при перегляді вимог і архітектури проектованої системи [9]

Виявити 100% помилок не представляється можливим ні одним з відомих методів, тому прогрес функціональної верифікації вимірятиметься виконанням пунктів складеного верифікаційного плану. Для кількісної оцінки повноти функціональної верифікації використовуються різні метрики

покриття: покриття коду, покриття функціональності, покриття станів і переходів кінцевих автоматів, покриття значень сигналів і переходів, покриття розгалужень тощо.

Методи функціональної верифікації поділяються на дві принципово різні групи:

– динамічні (тестові) методи, що вимагають наявності тестових (дій) сигналів, що аналізують реакції моделі;

– статичні (формальні) методи, які не потребують наявності тестових сигналів, орієнтовані на доказ досяжності очікуваних властивостей моделі.

У динамічних методах верифікації проект поміщається в середовище верифікації (TestBench), на вхід системи подаються тестові сигнали, що активують її, а вихідні реакції перевіряються на відповідність очікуваним еталонам. Тестові сигнали можуть бути приготовлені заздалегідь у вигляді кода TestBench, можуть бути прочитані з зовнішнього джерела, можуть бути тим чи іншим чином згенеровані під час моделювання.

Вхідні тестові дії, що активують конкретну функциональність або конкретну властивість системи, називаються спрямованими тестами. Оскільки такі тести стосуються простору станів системи що верифікується, про які відомо заздалегідь, істотна частина функціональних помилок знаходиться за межами цих обмежень. Для виявлення непокритих областей можуть бути згенеровані псевдовипадкові тестові набори (constrained random stimuli), які активують невідомі при спрямованих тестах стану системи. Такі згенеровані тестові сигнали не обмежуються вибором випадокових даних або адресів, а також орієнтуються на перебір керуючих послідовностей і комбінування зовнішніми по відношенню до системи проявами дій. Спрямовані і псевдовипадкові тести перевірки ефективні при спільному застосуванні - спрямованими тестами перевіряються основні функціональні вимоги, а псевдовипадковими - ймовірні відхилення від очікуваних умов.

Існує велика кількість типів моделей верифікації. Серед множини моделей особливо виділяють два типи: функціональні і часові. У першому

типі ігноруються часові характеристики проекту, в другому упор робиться на отримання оцінок часових властивостей проекту. Найважливішою характеристикою підходу є повнота охоплення проекту тестуванням. Методи оцінки повноти охоплення визначаються метрикою покриття досліджуваного пристрою тестами. Тестові послідовності для моделей, саму процедуру тестування і використовуване для цього устаткування називають TestBench. Верифікацію цього типу можна віднести до класу динамічної верифікації. Часто динамічну верифікацію ототожнюють з тестуванням за допомогою TestBench, хоча сфера застосування методів TestBench дещо вужчий.

Одним із способів істотного прискорення динамічної верифікації є застосування апаратної емуляції моделі. Проект розміщуються в перепрограмувальний FPGA-прототип, що розміщується в апаратному акселераторі, тим самим продуктивність верифікації збільшується на кілька порядків. Недоліком апаратної емуляції є висока вартість додаткових апаратних ресурсів в порівнянні з програмними варіантами, крім того, емулятори, як правило, не дозволяють отримати реалістичну картину моделювання з урахуванням затримок.

Інший ефективний спосіб прискорення верифікації – підвищення рівня абстракції. Очевидно, що верифікація на вентильному рівні (скалярна логіка управління, скалярні шляху даних) вимагає великих обчислюваненістю витрат, ніж верифікація на RTL-рівні (скалярна логіка управління, векторні шляху даних), а найбільш швидкої є верифікація на системному рівні із застосуванням об'єктно-орієнтованих підходів (TLM – транзакційний рівень моделювання).

Виявлені в результаті моделювання відхилення повинні бути усунені, після чого верифікація повторюється. Для обліку виявлених функціональних проблем використовуються баг-трекінгові системи, аналогічні системам, що застосовуються в програмної інженерії.

Таким чином можна сформулювати наступні етапи загального циклу проектування керуючих автоматів в системах логічного управління з

урахуванням процедур верифікації на різних етапах автоматизованого проектування..

1. Визначення технічних станів об’єкта управління, принципів роботи датчиків (перетворювачів) інформації про стан об’єкта управління, типи виконавчих механізмів на способи керування ними.

2. Визначення переліку вхідних та вихідних сигналів у кожному стані, систему подій, які впливають на функціонування об’єкта управління, часові параметри обробки вхідних сигналів у кожному стані та появи вихідних сигналів (затримки).

3. Формування технічного завдання (специфікації) на проектування пристрою керування.

4. Визначення типу пристрою керування, обрання моделі керуючого автомату та побудова графа переходів.

5. Перевірка графа на коректність у кожному стані шляхом аналізу повноти, несуперечності та можливості зациклювання у графі переходів.

6. Опис моделі пристрою керування на мові опису апаратури (мові програмування) та його автоматизоване проектування на обраній технологічній платформі (ПЛІС або мікроконтролер (МК)).

7. Функціональна верифікація розробленої HDL-моделі шляхом тестового моделювання та порівняння з еталонною специфікацією.

8. Синтез та імплементація

9. Верифікація (моделювання) синтезованої схеми у відповідний кристал ПЛІС з реальними затримками.

10. Макетування та фізичне моделювання розробленого прототипу пристрою керування у реальному часі.

2 ТЕХНОЛОГІЇ ВЕРИФІКАЦІЇ МОДЕЛЕЙ ЧАСОВИХ АВТОМАТІВ НА ПЛІС

2.1 Перевірка графових моделей керуючих автоматів на коректність

Серед усієї множини керуючих пристроїв можна виділити пристрої логічного управління, у яких управляючі сигнали (control value) представляються в двійковому алфавіті. Оскільки для реалізації керуючої частини в таких пристроях, як правило, використовуються кінцеві автомати, то ці автомати називаються керуючі автомати (КА). При описі алгоритму функціонування цифрових пристроїв логічного керування в САПР цифрових пристроїв одним із стилів написання коду на мовах опису апаратури (HDL-коду) є стиль автоматного програмування. Суть автоматного програмування полягає в відділенні опису логіки поведінки (за яких умов необхідно виконати ті чи інші дії) від опису його семантики (власне сенсу кожного з дій). Автоматні програми строго структуровані і в них виділено три види функцій: функції переходів, функції виходів та функції призначення нового стану. Автоматні програми також строго шаблонизовані з використанням операторів багатопозиційного вибору (switch, case) та умовних операторів (if, elsif-else). В якості алгоритмічної моделі для керуючих автоматів застосовується такий непроцедурний візуальний формалізм, як графи переходів, які в літературі також називаються діаграмами станів (State Diagrams). Зауважимо, що графи переходів автомата є не тільки візуальним відображенням алгоритму функціонування, але й повною його математичною моделлю. Предметом розгляду цього розділу є аналіз коректної побудови функцій умов переходів графових моделей керуючих автоматів, представлених у вигляді HDL-моделей синтезованої підмножини, с точки зору подальшого автоматизованого синтезу. Основні положення цього підходу викладені у роботі [10].

2.1.1 Ортогоналізація логічних функцій умов переходів

При аналізі надійності систем управління широко використовується поняття ортогональності, як системи несумісних подій. Поняття ортогоналізації також використовується для декомпозиції логічних функцій при синтезі цифрових систем та при перевірці графових моделей цифрових автоматів на коректність.

З точки зору реалізації алгоритму логічного управління структурний автомат це модель пристрою реального часу, яка характеризується видачею визначених вихідних сигналів в визначені моменти часу. Структурний автомат функціонує у автоматному часі, який вимірюється в тактах {t, t+1, t+2}, тобто автомат переходить з одного стану до іншого за один автоматний такт. За один автоматний такт автомат формує множину вихідних сигналів та обчислює значення наступного стану.

У аналітичному вигляді модель структурного автомату має вигляд $Y(t) = g(X(t), Z(t))$, $Z(t+1) = f(X(t), Z(t))$, де g – функція виходів структурного автомату, f – функція переходів структурного автомата. При цьому $Z(t+1) \equiv Z(t)$, але у наступному такті.

Таким чином, апаратна реалізація моделі структурного автомата (модель Хаффмена), складається з комбінаційного і послідовністного компонентів. Послідовністний компонент містить елементи пам'яті, такі як синхронні тригери, які запам'ятовують значення внутрішніх змінних (стан) і дозволяють змінювати його синхронно. Комбінаційний компонент складається з логічних елементів, які реалізують дві логічні функції: функцію виходів, яка обчислює значення вихідних сигналів, і функцію переходів, яка обчислює нові значення елементів пам'яті або внутрішніх змінних (тобто значення наступного стану). На рис.2.1 наведена графічна інтерпретація апаратної реалізації структурного автомату.

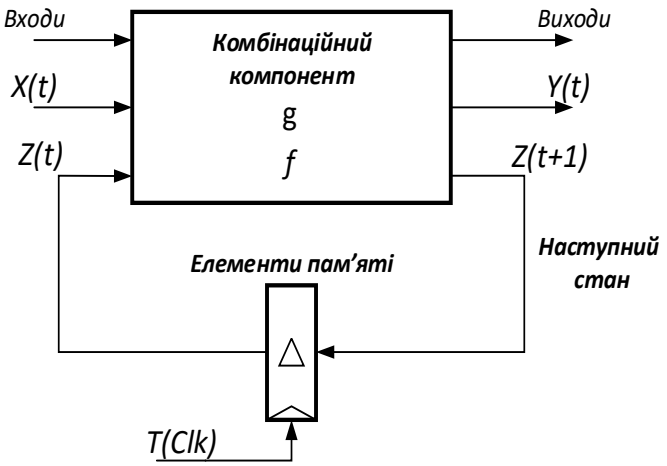

Рисунок 2.1 – Модель структурного автомату

В якості алгоритмічної моделі для керуючих автоматів застосовується такий непроцедурний візуальний формалізм, як графи переходів, які в англомовній літературі називаються діаграмами станів (State Diagrams). Зазначимо, що графи переходів є не тільки візуальним відображенням алгоритму функціонування автомату, але й повною його математичною моделлю [4].

Вершинам графа переходів відповідають стани автомата a_i, дугам відповідають переходи між станами. На дугах автомата моделі Мілі вказуються функції умов переходів $f_j(x_1, x_2, \ldots x_n)$ та вихідні (керуючі) сигнали. На дугах автомата моделі Мура вказуються тільки функції умов переходів, а вихідні сигнали вказуються біля вершин.

Кожен граф переходів має бути семантично і синтаксично коректний. Перша властивість визначає, чи правильна побудована модель, а друга - чи правильно вона побудована. При перевірці синтаксичної коректності граф переходів повинен перевіряться на досяжність, повноту, несуперечність і відсутність генеруючих контурів [1].

При перевірці синтаксичної коректності функцій умов переходів керуючого автомату вони повинні перевіряться на повноту та несуперечність.

На рис.2.2 наведений фрагмент графу переходів з вершиною a_i з якої виходять K дуг. Кожній дузі ставиться у відповідність логічний вираз функції

умов переходів $f(x_1,x_2,\dots x_n)$ у диз'юнктивній нормальній формі (ДНФ) :

$$f(x_1,x_2,\dots x_n) = f_1(x_1,x_2,\dots x_n) \vee \dots$$
$$\vee f_j(x_1,x_2,\dots x_n) \vee \dots \vee f_K(x_1,x_2,\dots x_n).$$

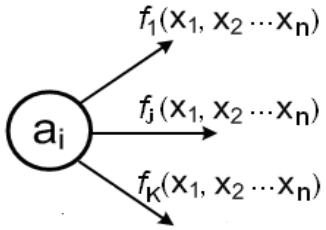

Рисунок 2.2 –. Відображення функцій умов переходів у графовій моделі

Повнота перевіряється для кожної вершини графа переходів шляхом аналізу умов переходів всіх дуг, які виходять з цієї вершини, тобто $\bigvee_{j=1}^{K} f_j(x_1,x_2,\dots x_n) = 1$. Повнота умов визначається як покриття всіх 2^n термів сукупності булевих функцій переходів, де n – кількість умов переходів (вхідних змінних, які ініціюють переходи з цієї вершини), тобто реалізується функція $f(x_1,x_2,\dots x_n) = 1$.

При забезпеченні несуперечливості для кожної вершини графа переходів перевіряється ортогоналізація булевих виразів умов переходів (відсутність спільних термів в булевих виразах умов для різних дуг) функцій переходів для дуг, які виходять з вершини, що розглядається, тобто $\forall(f_g \cdot f_h = 0)$, $g \neq h$.

Диз'юнктивна нормальна функція алгебри логіки називається ортогональною, якщо всі її кон'юнктивні терми взаємно ортогональні. Для такої ДНФ не існує набору значень її змінних, що входить більш ніж до однієї елементарної кон'юнкції, тобто на будь-якому наборі значень змінних значення одиниці може прийняти тільки одна кон'юнкція. Якщо логічну функцію представити у вигляді карті Карно, то їх зображення не будуть

перетинатися. Прикладом ортогональної ДНФ може служити її досконала форма (ДДНФ, рос. мовою СДНФ), що складається з повних взаємно ортогональних кон'юнкцій.

Розглянемо метод побудови ортогональної повної системи функцій переходів для вершини графа переходів.

Введемо наступні визначення: f – повна ДДНФ від n змінних, тобто $f(x_1,x_2,\dots x_n)=1$ – булева функція, яка приймає значення 1 на всіх 2^n наборах, f^* – повна ДДНФ від (n-1) змінних, f^{**} – повна ДДНФ від (n-2) змінних, f^{***} – повна ДДНФ від (n-3) змінних і так далі. Таким чином $f=f^*=f^{**}=f^{***}\equiv 1.$

Ортогональність термів булевої функції забезпечується за рахунок розкладання за відповідними змінними, з урахуванням повноти розкладанням за всіма змінними [7].

За першою теоремою розкладання:

$$f=\overline{x_1}\cdot f^*\vee x_1\cdot f^*=\overline{x_1}\cdot 1\vee x_1\cdot f^*=\overline{x_1}\vee x_1\cdot f^*=\overline{x_1}\vee$$
$$\vee x_1(\overline{x_2}\vee x_2\cdot f^{**})=\overline{x_1}\vee x_1(\overline{x_2}\vee x_2(\overline{x_3}\vee x\cdot f^{***}))=$$
$$=\overline{x_1}\vee x_1(\overline{x_2}\vee x_2(\overline{x_3}\vee x_3\cdot(\dots\cdot(\overline{x_n}\vee x_n)))).$$

Таким чином, повна булева функція від n змінних розкладається мінімум на (n+1) кон’юнкцій при збереженні суттєвості всіх n змінних.

В якості прикладу розглянемо повну ДДНФ від трьох змінних $f(x_1, x_2, x_3)=1$. За визначенням ДДНФ є ортогональною. Запишемо повну ДДНФ та виконаємо розкладення по x_1 з заміною лівої частини розкладення у дужках на 1.

$$f(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}\,\overline{x_3}\vee\overline{x_1}\,\overline{x_2}x_3\vee\overline{x_1}x_2\overline{x_3}\vee$$
$$\vee\overline{x_1}x_2x_3\vee x_1\overline{x_2}\,\overline{x_3}\vee x_1\overline{x_2}x_3\vee x_1x_2\overline{x_3}\vee x_1x_2x_3=$$
$$=\overline{x_1}(\overline{x_2}\,\overline{x_3}\vee\overline{x_2}x_3\vee x_2\overline{x_3}\vee x_2x_3)\vee x_1(\overline{x_2}\,\overline{x_3}\vee$$
$$\vee\overline{x_2}x_3\vee x_2\overline{x_3}\vee x_2x_3)=\overline{x_1}\cdot 1\vee x_1\cdot(\overline{x_2}\,\overline{x_3}\vee\overline{x_2}x_3\vee$$
$$\vee x_2\overline{x_3}\vee x_2x_3)=\overline{x_1}\vee x_1(\overline{x_2}\,\overline{x_3}\vee\overline{x_2}x_3\vee x_2\overline{x_3}\vee x_2x_3).$$

Виконаємо аналогічно процедуру розкладення по x_2 для виразу у дужках. Розкриємо дужки та отримаємо повну ортогональну функцію від трьох змінних:

$$\begin{aligned} &f(x_1,x_2,x_3) = \overline{x_1} \vee x_1(\overline{x_2}(x_3 \vee \overline{x_3}) \vee x_2(\overline{x_3} \vee x_3)) = \\ &= \overline{x_1} \vee x_1(\overline{x_2} \cdot 1 \vee x_2(\overline{x_3} \vee x_3)) = \overline{x_1} \vee x_1(\overline{x_2} \vee x_2\overline{x_3} \vee \\ &\vee x_2x_3) = \overline{x_1} \vee x_1\overline{x_2} \vee x_1x_2\overline{x_3} \vee x_1x_2x_3 . \end{aligned} \qquad (2.1)$$

Виходячи з цього можна зробити висновок, що повна ортогональна функція від n змінних має в своєму складі не менше, ніж (n+1) кон'юнкцій. Отже, з кожного стану автомату з функцією умов переходів від n змінних $f(x_1,x_2,\dots x_n)$ повинно бути не менше, ніж (n+1) переходів.

Одним із способів візуального аналізу ортогональності функцій переходів є представлення ортогональних функцій на картах Карно. На рис.2.3 (а) зображена карта Карно для ортогональної функції (1). З наведеної карти видно, що для ортогональної функції обведення груп одиниць для повної ДДНФ не перетинаються, тобто кон'юнкції не мають спільних частин.

На рис. 2.3 (б) представлена карта Карно для ортогональної функції $f(x_1,x_2,x_3) = \overline{x_1} \vee x_1\overline{x_2} \vee x_1x_2\overline{x_3}$, але наведена функція не є повною, бо в ній відсутнє обведення $x_1x_2x_3$. Це обумовлено тим, що при побудові вказаної функції порушено правило повноти функції, тобто $f^* \neq 1$.

На рис. 2.3(в) також представлена карта Карно для ортогональної функції $f(x_1,x_2,x_3) = \overline{x_1} \vee x_1\overline{x_2} \vee x_1x_2$, але наведена функція теж не є повною, бо в ній відсутня змінна x_3. Це обумовлено тим, що при побудові вказаної функції порушено правило наявності у повній не менше, ніж функції (n+1) кон'юнкцій.

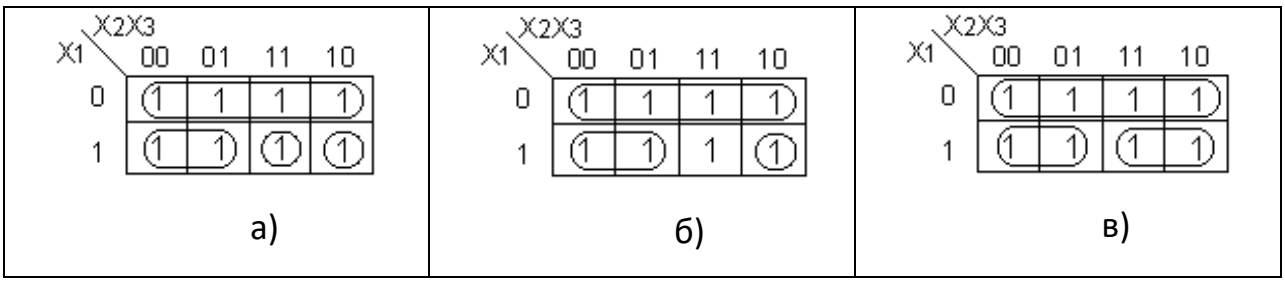

Рисунок 2.3 – Карти Карно для ортогональних функцій

Ще одним способом візуального аналізу ортогоналізації повних булевих функцій є використання бінарних дерев рішень у формі граф-схем алгоритмів (ГСА) (рис. 2.4).

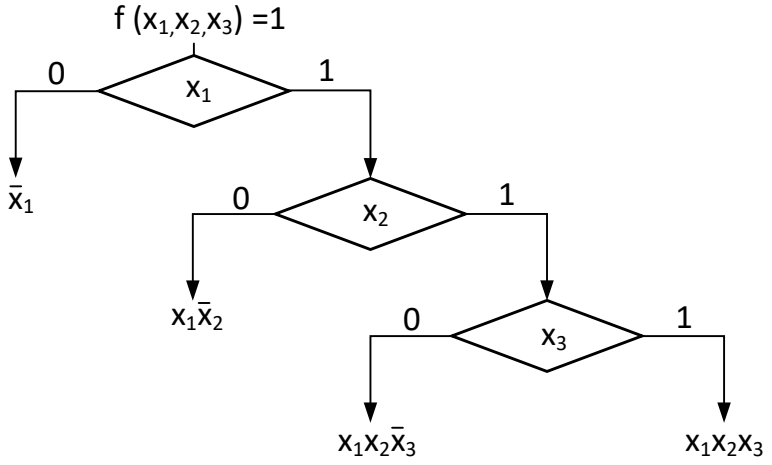

Рисунок 2.4 – Дерево ортогональних рішень

Дерево ортогональних рішень на цьому рисунку відповідає карті Карно, яка зображена на рис.2.3,(а). Тут 3 умовних вершини та 4 переходи (шляхи розалуження), умови переходу по кожній з гілок відповідають термам ортогональної ДНФ (2.1).

При представленні функції умов переходів бінарним деревом ортогональних рішень можна точно визначити необхідне число переходів. При незалежності умов $x_1, x_2, \ldots x_n$ одної від одної це число буде дорівнювати (m + 1), де m – число вершин дерева (але не змінних, так як змінні у вершинах таких дерев можуть повторюватися).

2.1.2 Аналіз коректності HDL-моделей керуючих автоматів при автоматизованому синтезі

У цьому пункті приведена ілюстрація запропонованих теоретичних викладень на прикладах графових моделей автоматів, їх опису на мові VHDL, результатів їх моделювання та синтезу.

Розглянемо приклад графа переходів (рис.2.5) з коректними умовами переходів із стану a_1 відповідно до дерева ортогональних рішень, наведеному на рис.2.4. Функція умов переходів із стану a_1 ортогональна, отже, умови переходів несуперечливі. Крім того, функція є повною. VHDL-модель цього автомата наведена на рис.2.6.

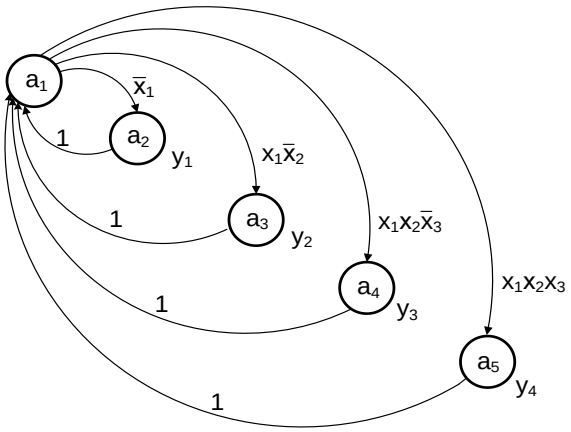

Рисунок 2.5 – Граф переходів автомата Мура з коректними умовами переходів

```
library IEEE;
use IEEE.std_logic_1164.all;
entity Fsm_right is
        port (x1, x2, x3, Clk, reset: in STD_LOGIC;
        y1, y2, y3, y4: out STD_LOGIC);
end;

architecture Fsm_right of Fsm_right is
        type State_type is (a1, a2, a3, a4, a5);
        signal State, NextState: State_type;
begin
```

```
Sreg0_CurrentState: process (Clk, reset)
begin
        if reset='1' then State <= a1;
        elsif Clk'event and Clk = '1'
        then State <= NextState;
        end if;
end process;
Sreg0_NextState: process (State, x1, x2, x3)
begin
case State is
when  a1=> if x1='0' then NextState <= a2;
        elsif x2='0' then NextState <= a3;
        elsif x3='0' then NextState <= a4;
        else NextState <= a5;
        end if;
when  a2=>  NextState <= a1;
when  a3=>  NextState <= a1;
when  a4=> NextState <= a1;
when  a5=> NextState <= a1;
when others => NextState <= a1;
end case;
end process;
y1 <= '1' when State=a2 else '0';
y2 <= '1' when State=a3 else '0';
y3 <= '1' when State=a4 else '0';
y4 <= '1' when State=a5 else '0';
end;
```

Рисунок 2.6 – VHDL-модель автомата Мура з коректними умовами переходів

Часові діаграми роботи автомата наведені на рис.2.7. На ньому відображені результати моделювання в системі Active-HDL ALDEC на усіх комбінаціях наборів умов x_1, x_2, x_3. Діаграма свідчить, що функція умов переходів є повною та вона ортогональна і повністю повторює вприклад з рис.4. У період з150 нс по 950 нс автомат переходить у стан a_2 $(y_1 = 1)$, доки виповнюється умова $\bar{x}_1$, тобто $(x_1 = 0)$ та повертається зворотно в a_1 $(y_1 = 0)$. В період з 950 нс по 1350 нс автомат переходить в стан a_3 $(y_2 = 1)$, доки виконується умова $x_1\bar{x}_2$, тобто $(x_1 = 1, x_2 = 0)$ та повертається зворотно в a_1 $(y_2 = 0)$. У період з 1350 нс по 1550 нс автомат переходить в стан a_4 $(y_3 = 1)$, доки виконується умова $x_1x_2\bar{x}_3$, тобто $(x_1 = 1, x_2 = 1, x_3 = 0)$ та повертається зворотно в a_1 $(y_3 = 0)$. У період з 1550 нс по 1750 нс автомат переходить в

стан a_5 $(y_4 = 1)$, доки виконується умова $x_1x_2x_3$, тобто $(x_1 = 1, x_2 = 1, x_3 = 1)$ та повертається назад в a_1 $(y_4 = 0)$.

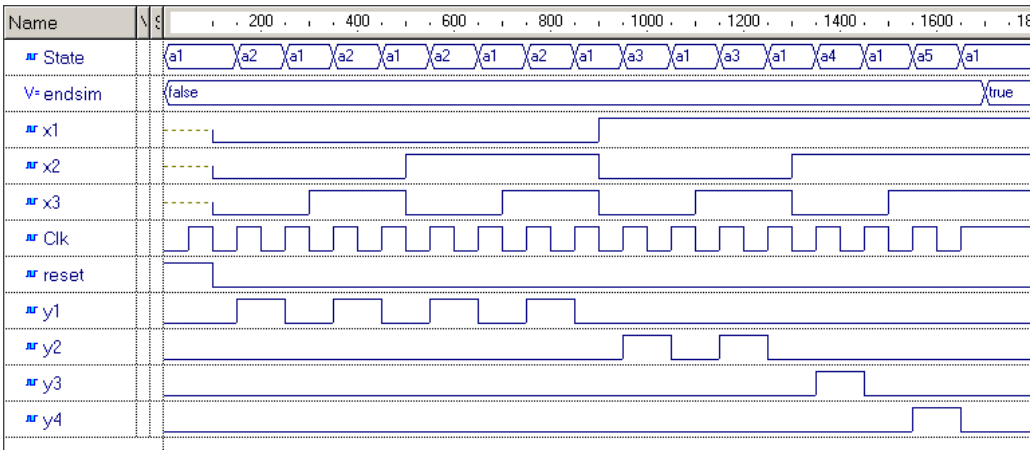

Рисунок 2.7 – Часові діаграми автомата Мура з коректними умовами переходів

Розглянемо наступний приклад графа переходів (рис.2.8). У ньому умови переходів із стану a_1 некоректні з точки зору ортогоналізації функції умов переходів, при переході в a_4 і a_5 в записі термів бракує змінної x_1, але вони не суперечать умовам переходів $x_1x_2\bar{x}_3$ та $x_1x_2x_3$. Функції умов переходів є повними.

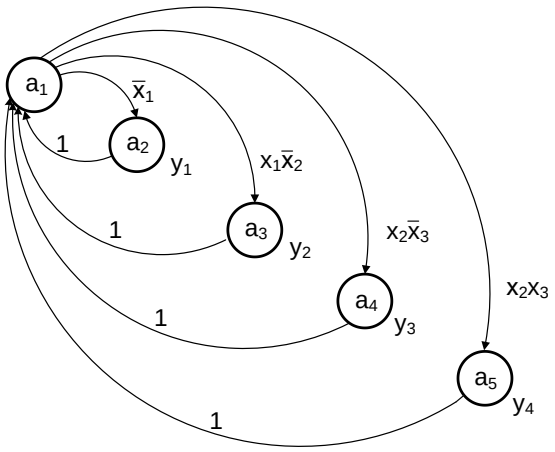

Рисунок 2.8 – Граф переходів автомата Мура з несуперечливими неповними умовами переходів

Переходи із стану a_1 можуть бути записані на мові VHDL таким чином (рис.2.9, а або рис.2.9, б). При цьому такий опис стилістично не є коректним, але не є суперечливим та при моделюванні дає такі ж результати, що і на рис.2.7.

```
when  a1=> if x1='0' then NextState <= a2;
      elsif x1='1'and x2='0' then NextState <= a3;
      elsif x2='1'and x3='0' then NextState <= a4;
      else NextState <= a5;
      end if;
```

а)

```
when  a1=> if x1='0' then NextState <= a2;
      elsif x1='1'and x2='0' then NextState <= a3;
      elsif x2='1'and x3='0' then NextState <= a4;
      elsif x2='1'and x3='1' then NextState <= a5;
      end if;
```

б)

Рисунок 2.9 – Фрагменти VHDL- моделі автомата Мура з несуперечливими неповними умовами переходів

Крім того, моделі автоматів, що наведені на рис.2.6 та 2.9, при синтезі дають абсолютно однакові коректні результати. Синтез виконувався в системі XILINX ISE.

Далі розглянемо приклад графа переходів (рис. 2.10) з відсутнім переходом (по умові $x_1x_2x_3$) та неповною умовою переходу із стану a_1 в стан a_4: $x_2\bar{x}_3$ замість $x_1x_2\bar{x}_3$. Функція умов переходів в даному випадку є не ортогональною та неповною. Фрагмент VHDL- моделі цього автомата наведений на рис. 2.11.

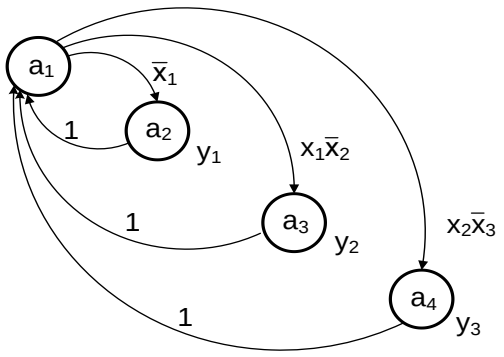

Рисунок 2.10 – Граф переходів автомата Мура з відсутнім переходом

```
when  a1=> if x1='0' then NextState <= a2;
           elsif x1='1'and x2='0' then NextState <= a3;
          elsif x2='1'and x3='0' then NextState <= a4;
          end if;
```

Рисунок 2.10 – Фрагмент VHDL-моделі автомата Мура з відсутнім переходом

При моделюванні роботи даного автомата (рис. 2.12) на перший погляд все добре, але насправді змінна x_3 тут виявляється несуттєвою, як на рис. 3,(в), на наборі x_1, x_2, x_3=111 автомат не має переходити в жодний стан, але система моделювання переводить його в стан a_4. Так само при значеннях умов x_1, x_2, x_3= 010, автомат може перейти в два стани a_2 і a_4, чого не повинно бути при правильній роботі автомата, але система моделювання маскує таку ситуацію, переводячи автомат в стан a_2. У цьому прикладі є і відсутність повноти, і наявність суперечності умов переходів, але на етапах аналізу синтаксису і моделювання це явно не проявляється.

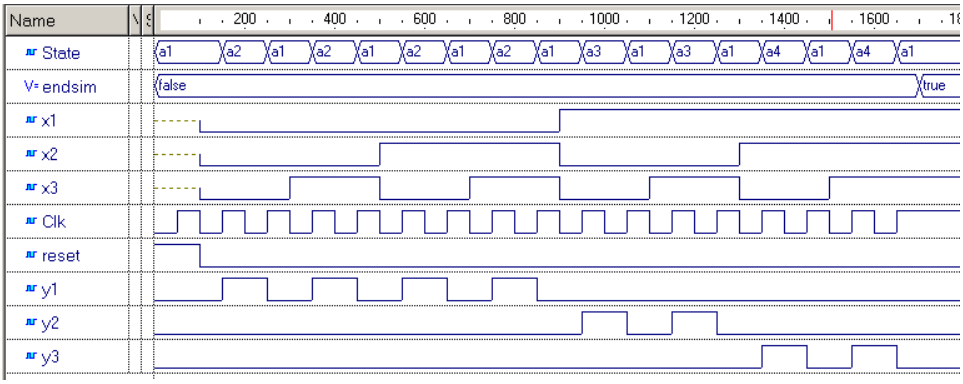

Рисунок 2.12 – Часові діаграми автомата Мура з відсутнім переходом

При синтезі цього автомата виникають проблеми. Критичної помилки не виникає, але з'являється попередження про появу чотирьох тригерів типу latch на додаток до чотирьох тригерів типу flip-flop: WARNING: Xst:737 – Found 4-bit latch for signal <NextState>. Latches may be generated from incomplete case or if statements. We do not recommend the use of latches in FPGA/CPLD designs, as they may lead to timing problems. Тобто замість двох

тригерів для чотирьох станів синтезується 8 тригерів двох типів. Цього не повинно бути в коректно синтезованому автоматі.

На завершення розглянемо приклад графа переходів (рис. 2.13) з суперечливою умовою переходу із стану a_1 в стан a_4: $\bar{x}_2 x_3$ замість $x_1 x_2 \bar{x}_3$. Фрагмент VHDL- моделі цього автомату наведена на рис. 2.14.

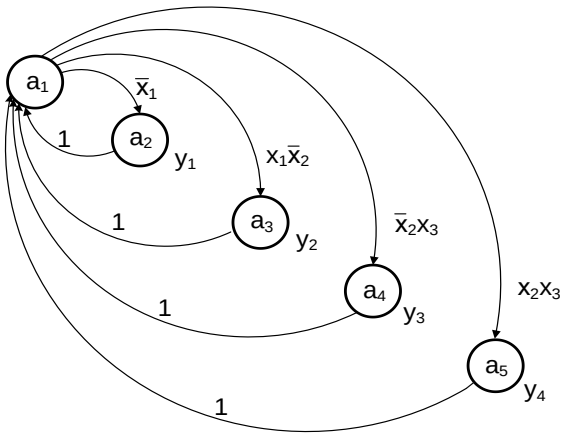

Рисунок 2.13 – Граф переходів автомата Мура з суперечливою умовою переходу

```
when  a1=> if x1='0' then NextState <= a2;
        elsif x1='1' and x2='0' then NextState <= a3;
        elsif  x2='0' and x3='1' then NextState <= a4;
        elsif  x2='1'and x3='1' then NextState <= a5;
        end if;
```

Рисунок 2.14 –. Фрагмент VHDL-моделі автомата Мура з суперечливою умовою переходу

При моделюванні роботи цього автомата (рис. 2.15) явно спостерігається тільки неможливість переходу в стан a_4 ні при яких наборах умов x_1, x_2, x_3. Як і в попередньому прикладі система моделювання маскує помилкову ситуацію, при якій створюються умови переходу автомата в декілька станів, наприклад, при наборі умов $x_1, x_2, x_3 = 001$ автомат повинен перейти в два стани a_2 і a_4, при наборі умов $x_1, x_2, x_3 = 101$ автомат повинен перейти в a_3 і a_4, а також помилкову ситуацію, при якій створюються умови,

при якій автомат не повинен переходити ні в жоден стан, наприклад, при набір умов $x_1, x_2, x_3 = 110$. У даному прикладі також є і відсутність повноти, та наявність суперечності умов переходів, але на етапах аналізу синтаксису це ніяк не проявляється, а на етапі моделювання явно видно тільки те, що автомат ніколи не переходить в стан a_4.

При синтезі теж виникають проблеми. З'являється попередження про появу п'яти тригерів типу latch на додаток до п'яти тригерів типу flip-flop:Xst:737 - Found 5-bit latch for signal <NextState>. Latches may be generated from incomplete case or if statements.

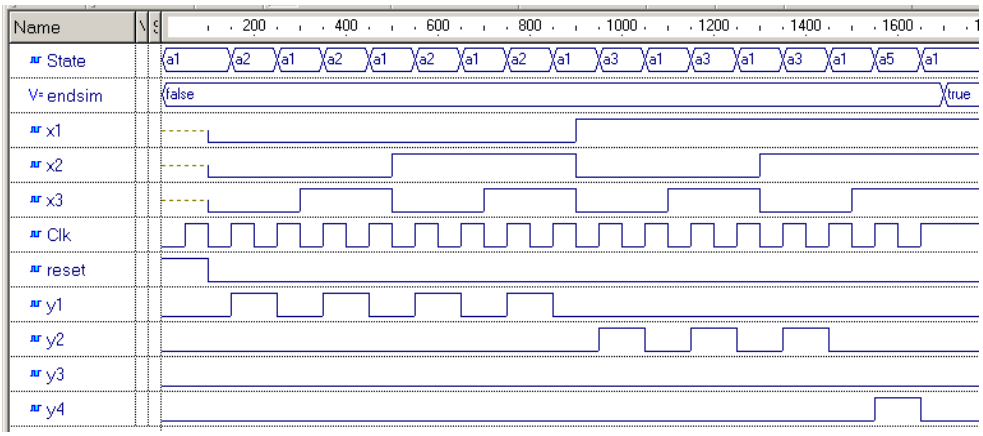

Рисунок 2.15– Часові діаграми автомата Мура з суперечливою умовою переходу

Таким чином показано, що проблеми, пов'язані з некоректними умовами, можуть проявитися тільки в процесі синтезу, тому перевірку умов переходів на несуперечність і повноту необхідно виконувати ще на етапі формування моделі графу переходів автомата.

В результаті проведених досліджень було показано наступне.

1. Функція умов переходів $f(x_1,x_2,\dots x_n)$ несуперечлива, якщо вона ортогональна.

2. Ортогональна функція умов переходів $f(x_1,x_2,\dots x_n)$ є повною, якщо її терми покривають усі набори $x_1,x_2,\dots x_n$.

3. Число переходів із стану a_i з функцією умов переходів $f(x_1,x_2,...x_n)$ має бути не менше (n+1), де n – число змінних.

4. Число переходів із стану a_i з функцією умов переходів має бути рівне (m+1), де m – число вершин дерева ортогональних рішень.

5. При невиконанні п. 1-4 висновків синтезується автомат з надмірним числом тригерів типу flip-flop та неприпустимою появою такого ж числа тригерів типу latch.

2.2 Технології функціональної верифікації в САПР XLINX ISE

Верифікація HDL-проекту - це процес доказу того, що він функціонує згідно його специфікації. Традиційний підхід, так звана імітаційна верифікація, припускає моделювання HDL-опису об'єкту, що проектується і відтворення в ході імітаційного експерименту усіх (чи частини) можливих вхідних дій і внутрішніх станів модельованої системи, що верифікується [9].

При верифікації систем логічного управління є два аспекти: що перевіряти та як перевіряти.

Перший аспект обумовлює створення контенту верифікації, а саме стани технічної системи, вихідні сигнали часові параметри функціонування системи у реальному часі, робочі та аварійні стани системи управління, «виключні» події та дії (які фізично або технічно не можуть реалізовуватися одночасно), принципи аналізу повноти верифікації або умови її ресурсного припинення.

Другий аспект обумовлює спосіб створення середовища верифікації, тобто програмно апаратних засобів реалізації першого аспекту. Тут слід враховувати технологічну платформу реалізації як самої системи логічного управління так і програмно-апаратних засобів реалізації засобів середовища верифікації. Середовища верифікації повинно мати наступні властивості (реалізувати наступні функціональні можливості):

– спосіб збереження та подачі вхідних сигналів на модель пристрою;

– спосіб збереження еталонів;

– спосіб порівняння вихідних сигналів з еталоном;

– спосіб відображення та обробки результатів верифікації;

– набор стандартних (алгоритмічних) тестових генераторів;

– способи синхронізації з пристроєм, що верифікується (моделлю);

– спосіб технічної реалізації середовища верифікації разом з пристроєм, що верифікується.

Засобами верифікації можуть бути різні середовища САПР. Наприклад при реалізації пристрою керування на мікроконтролерній технологічній платформі це може бути САПР Proteus. При реалізації пристрою керування на технологічній платформі ПЛІС це можуть бути САПР Xilinx ISE. Active-HDL тощо.

Процес верифікації починається з розробки тест-плану (як передбачається тестувати, яким інструментарієм, в якому порядку і коли, прості і граничні тести, заборонені режими тощо).

Потім слідує:

– створення середовища верифікації (TestBench);

– відладка програми тестування і моделі досліджуваного об'єкту;

– профілізація (виявлення і по можливості усунення ділянок тіста, що сильно знижують швидкодію);

– проведення регресійних (багатократних, вичерпний) експериментів.

У свою чергу тест - це набір зовнішніх сигналів, що подаються на об'єкт, що перевіряється. Він має бути:

– повним, тобто виявляти усі можливі помилки проекту (в даному випадку функціональні);

– коректним, тобто не містити заборонені комбінації вхідних сигналів;

– компактним, тобто перевіряти тестований об'єкт за прийнятний машинний час, і так далі

TestBench є спеціалізованою програмою, яка в автоматичному режимі подає на HDL-модель, що верифікується (UUT - Unit Under Test), тестові дії (stimuli) і перевіряють коректність видаваних нею відповідей (рис. 2.16) [8].

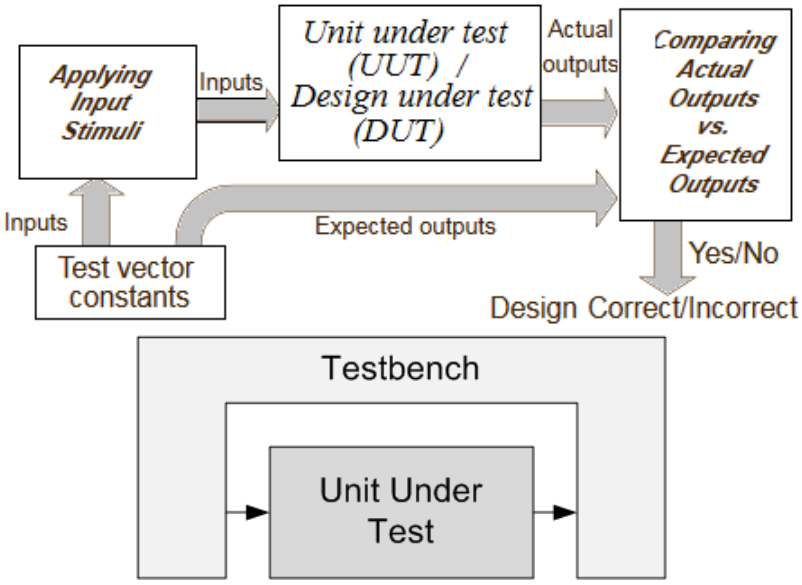

Рисунок 2.16 – Тестове середовище Testbench

Іншими словами, VHDL TestBench – це середовище верифікації або «віртуальний стенд тестування» вашої схеми. Ви створюєте середовище, усередині якого вставляєте вашу схему як компонент. Усередині цього середовища ви генеруєте вхідні сигнали для вашої схеми, подаєте їх на її входи, щоб потім проаналізувати її виходи. При цьому сама TestBench не має зовнішнього інтерфейсу.

Цей файл можна промоделювати в спеціальній програмі симуляторі, яка виконуватиме дії, описані в цьому файлі і представляти результат в графічному виді у вигляді часової діаграми сигналів (waveform). Змінюваним параметром буде час – симулятор послідовно збільшуватиме час з необхідним дозволом і послідовно обчислюватиме значення сигналів.

TestBench - це такий же VHDL-файл, компонентом якого є ваша схема, описана на VHDL. Тому він має стандартну структуру VHDL -описания, яка включає наступні елементи:

– посилання на використовувані бібліотеки і пакети;

– опис інтерфейсу об'єкту (entity);

– опис архітектури об'єкту;

– декларація компонента, що представляє модуль опису верхнього рівня ієрархії проектованого пристрою;

– декларація сигналів, використовуваних для подачі тестових дій і контролю вихідних реакцій модельованої системи;

– вираження створення екземпляра компонента з підключенням відповідних сигналів;

– формування поведінки вхідних сигналів.

Приклад мовного шаблону TestBench наведений на рис. 2.17

```
entity testbench_ent is
end entity testbench_ent;

architecture testbench_arch of testbench_ent is
  signal declarations
  component declarations

begin
  component instantiations
  stimuli (test vectors)
end architecture testbench_arch;
```

Рисунок 2.17 – Шаблон TestBench

Під об'єктом опису розуміється модель «випробувального стенду» в цілому, тому він не має інтерфейсних сигналів (портів). Архітектура цього об'єкту містить єдиний компонент, що представляє пристрій, що розробляється.

Мова VHDL допускає вбудовування у файл Testbench перевірку виконання певних умов. Це виробляється за допомогою оператора assert. Приклад його використання показаний нижче:

```
assert SUM_EXPECTED = SUM_OUT        -- условие для проверки
report "ERROR: output SUM is incorrect" severity warning;
```

Після ключового слова assert вказується умова, яка передбачається істинною (при нормальній роботі модуля воно повинне виконуватися). Якщо умова не виконується, в консоль виводиться повідомлення, вказане після ключового слова report. Потім severity вказується рівень важливості умови, що перевіряється.

Рівень важливості може бути наступним: note – "примітка", це повідомлення не є помилкою і носить інформаційний характер;

warning – "попередження", такі повідомлення підраховуються, але їх наявність не є основою для визнання результатів некоректними;

error – "помилка", результат моделювання визнається некоректним, моделювання триває;

failure – "збій", аналогічно помилці, але виконання негайно уривається. За умовчанням повідомлення мають рівень error.

Призначення рівня важливості виробляється самим розробником тесту. Передбачається, що повідомлення рівня "збій" ставиться у відповідність помилкам, які не можуть бути досить легко виправлені, або відносяться до "ситуацій, які ніколи не можуть виникнути". Виникнення повідомлення такого рівня повинне означати, що в процесі моделювання виникли істотні проблеми, і виробляти подальший аналіз не має сенсу.

Розглянемо процес створення Testbench в САПР Xilinx ISE WebPack 14.7 і запуску його на моделювання з використанням вбудованого ISIM - симулятора.

Для додавання файлу VHDL тестових дій необхідно виконати наступні дії:

– запустити майстер створення файлів проекту шляхом вибору "New Source". у меню пункт "Project";

– як тип нового файлу вибрати "VHDL Test Bench";

– ввести ім'я файлу в поле "File name";

– переконатися, що опція "Add to project" включена. Натиснути на кнопку "Next".

На другому кроці роботи майстра створення файлів проекту необхідно вибрати файл проекту, з яким він буде асоційований. Після вибору файлу необхідно натиснути на кнопку "Next".

На останньому кроці роботи майстра приводиться короткий звіт по роботі. По натисненню кнопки "Finish" майстер створює файл VHDL тестової дії з вказаними властивостями і завершує свою роботу.

Редагування VHDL файлу тестових дій здійснюється в текстовому редакторові і аналогічно редагуванню VHDL коду.

Після створення і підготовки файлу тестових дій необхідно запустити процес моделювання роботи. Для цього необхідно виконати наступне.

Перейти в режим "Simulation" у вікні опису проектів ("Source window"). У випадному списку, показаному на рис.2.18, вибрати режим моделювання роботи.

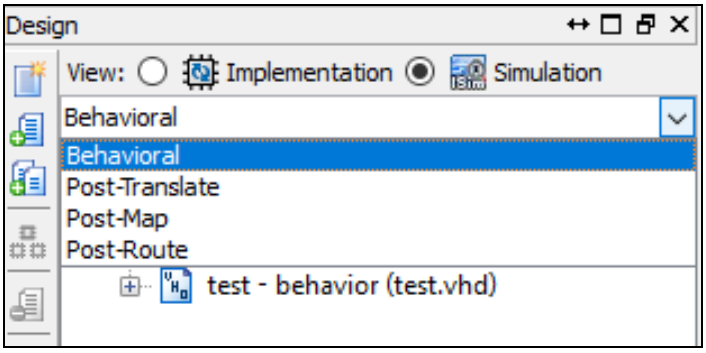

Рисунок 2.18 – Режими моделювання

Режими моделювання роботи ПЛИС:

"Behavioral" - функціональне моделювання;

"Post-Translate" - моделювання після прив'язки до ПЛИС;

"Post-Map" - моделювання після розміщення в ПЛИС;

"Post-Route" - моделювання після трасування сигналів в ПЛИС з урахуванням усіх затримок.

Виділити файл тестової дії у вікні опису проектів ("Source window").

Для першого запуску моделювання з САПР Xilinx ISE WebPACK 14.7, необхідно зробити налаштування параметрів моделювання роботи. Якщо усе вже налагоджено - необхідно перейти до кроку 7.

Вибрати "Process Properties" в контекстному меню, що з'являється при правому клацанні миші по "Simulate . Model", як показано на рис. 2.19.

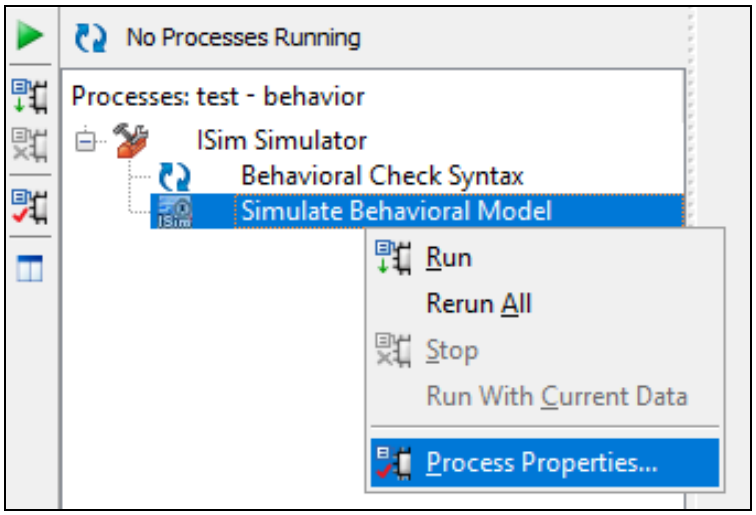

Рисунок 2.19 – Відкриття вікна налаштування симулятора

Зовнішній вигляд вікна налаштування симулятора показаний на рис 5.

Основними параметрами налаштування симулятора є:

"Run for Specified Time" - опція дозволу вказівки часу моделювання роботи;

"Simulation Run Time" - час моделювання роботи;

"ISim UUT Instance Name" - назва тестованого модуля у файлі тестової дії.

Детальніше про інших властивості проекту можна шанувати, натиснувши на кнопку "Help".

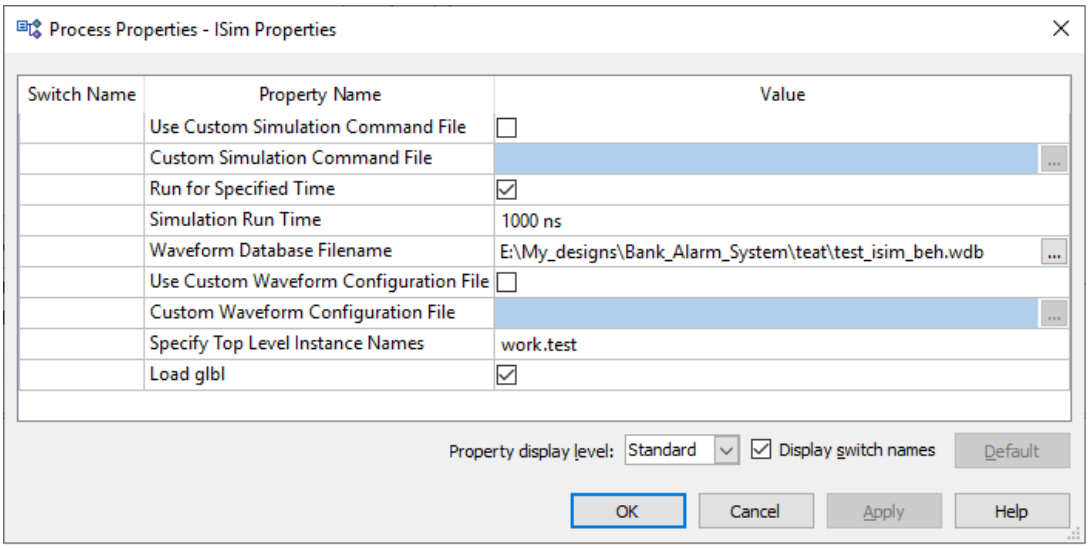

Рисунок 2.20 – Вікно налаштування симулятора

Подвійним клацанням миші по "Simulate . Model", що знаходиться у вікні процесів в розділі "ISim Simulator" (рис. 2.21), де під "". розуміється вибраний режим моделювання роботи, запускається симулятор ISim.

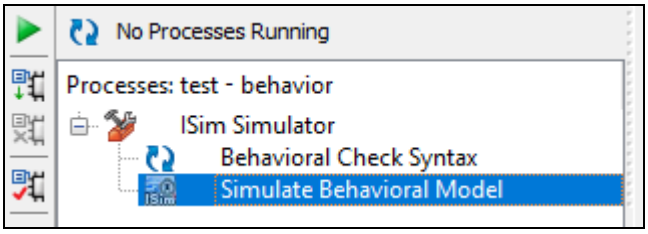

Рисунок 2.21 – Запуск TestBench на моделювання

Результати моделювання представляться у формі часової діаграми Waveform (рис. 2.22).

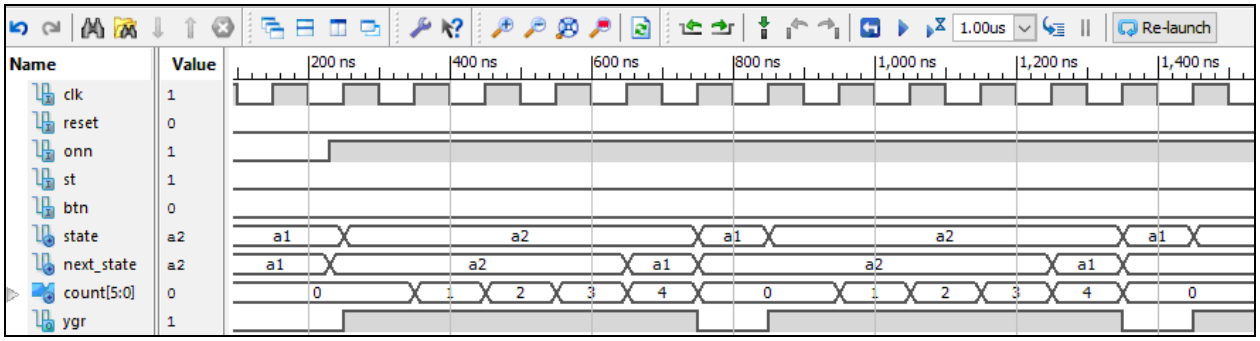

Рисунок 2.22 – Приклад Waveform

2.3 Апаратна верифікація проектів керуючих автоматів в ПЛІС

Для збільшення швидкості моделювання на кілька порядків застосовуються апаратні прискорювачі, або платформи емуляції [11]. Представлені на ринку прискорювачі реалізовані на спеціальних процесорах, на замовних або на комерційних ПЛІС (табл. 1). Всі три архітектури є масштабованими, вони дозволяють розміщувати проекти будь-якого розміру, від IP-блоків до повних систем. Важливим є тестування взаємодії на системному рівні, підтримуване апаратними прискорювачами. Частота залежить від багатьох чинників і становить від одиниць МГц до 100 МГц (для HAPS-80). Зазвичай підтримуються методологія UVM, верифікація на основі тверджень, розрахована на багато користувачів робота, моделі рівня транзакцій (TLM).

Сучасні SoC можуть включати в себе кілька процесорних ядер і високопродуктивні інтерфейси. Для управління використовуються операційна система і драйвери.

Таблиця 2.1 – Апаратні прискорювачі і платформи для емуляції

Платформа	Компанія	Мікросхеми	Кількість вентилів у проекті
Palladium	Cadence	Спеціалізовані процесори	До 2 млрд
Protium	Cadence	ПЛИС Xilinx Virtex-7	До 100 млн
Veloce II	Mentor	Спеціалізовані ПЛИС Crystal2	От 256 млн до 2 млрд
ZeBu Server-3	Synopsys	ПЛИС Xilinx Virtex-7	300 млн, масшта-бується до 3 млрд
HAPS-80	Synopsys	ПЛИС Xilinx Virtex UltraScale	До 1,6 млрд
Prodigy Cloud Cube 32	S2C	ПЛИС Xilinx Virtex-7 (UltraScale, Kintex-7) або Altera Stratix IV	До 1,4 млрд

Емуляція на основі процесорів реалізована в прискорювачах сімейства Palladium. Апаратна частина емулятора складається з масиву спеціалізованих процесорів, що виконують булеві операції проекту, що тестується.. Емулятор включає в себе від 103 до 104 арифметико-логічних пристроїв. Завдання ПЗ полягає в поділі проекту між процесорами і в оптимальному плануванні булевих операцій в коректній часовій послідовності. Підтримка стандартних інтерфейсів реалізована за рахунок SpeedBridges - карт адаптації частот емулятора і периферійних пристроїв.

Основними перевагами Palladium є швидкий час компіляції і повна видимість проекту на повній швидкості емуляції. Palladium ефективний в режимі ICE, що підтримується набором карт SpeedBridges. Є можливість оцінки обсягу живлення проекту з метою зменшення енергоспоживання. Головні недоліки Palladium – великі фізичні розміри і високе енергоспоживання в порівнянні з ПЛІС-емулятором еквівалентної обчислювальної потужності. Обмежена швидкість в режимі Transaction-based acceleration. Також слід зазначити високу вартість апаратних емуляторів.

Прискорювачі на основі спеціалізованих ПЛІС забезпечують стовідсотковий доступ без компіляції пробників і швидку трасування часових діаграм. Недоліком є використання системи робочих станцій для швидкої компіляції, швидкість яких нижче, а фізичні розміри більше, ніж у еквівалентного емулятора на основі комерційних ПЛІС.

Прискорювачі на основі комерційних ПЛІС володіють найменшими фізичними розмірами і споживаної потужністю. Вони досягають більш високої швидкості виконання. Недоліки полягають в меншій швидкості компіляції в порівнянні з іншими двома архітектурою, по крайней мере, для проектів в 10 млн логічних елементів або менше. Повна видимість проекту досягається в обмін на високу швидкість емуляції.

Cadence повідомляє про використання Palladium XP II компанією Realtek для розробки і верифікації проекту SoC. Відзначаються прискорення

в 250 разів у порівнянні з попередньою методологією, поліпшення якості верифікації, економія часу за рахунок повторного використання більш ніж 90% попередніх налаштувань середовища моделювання, а також оцінка споживаної потужності в межах 5% від фактично виміряної потужності SoC.

Palladium XP використовувався компанією Altair Semiconductor (викуплена Sony в 2016 р) для перевірки та затвердження проекту Internet of Thing. Повідомляється про зменшення часу циклу розробки на 20% Виконано розробка ПО і його перевірка в апаратній контексті.

Компанія NVIDA застосовувала установку, що складається з 16 шасі, під назвою Tigris (Cadence) для емуляції архітектури GPU Fermi. Найбільша установка Palladium XP під назвою Indus використовувалася для емуляції архітектури наступного покоління Kepler].

Компанія AMD використовувала ZeBy для емуляції графічного процесора (GPU), імеющего240 млн вентилів. Було задіяно від 50 до 70 ПЛІС. Застосовано підхід, заснований на транзакціях. Перевірялося функціонування GPU в середовищі IBM PC-сумісної ЕОМ. Різні материнські плати IBM-PC моделювалися за допомогою віртуальної машини VirtualBox. Було зроблене завантаження ОС Linux, драйвера, виконувалися операції читання / запису пам'яті емульованого GPU. Також були виконані 20 базових апаратних тестів для GPU.

Платформа емуляції Veloce була використана Barefoot Networks для перевірки високопродуктивного мережевого комутатора Tofino 6.5 ТБ / сек. Важливим було використання інструменту Veloce Virtual LAB Ethernet, що включає в себе генератор і монітор пакетів Ethernet (EPGM).

HAPS-70 використовувався для верифікації двоядерного процесора Baikal-T1 компанії «Байкал електронікс». Мікропроцесор працює на частоті 1,2 ГГц, використовує 32-бітові ядра MIPS Warrior P5600, має в своєму складі ряд високошвидкісних периферійних пристроїв, таких як SATA, PCI Express, і Ethernet 10 Гбіт / с. Тестування проводилося в кілька етапів. Повідомляється про завантаження ОС Linux і запуску одного ядра на частоті 25 МГц.

Перевагами апаратних прискорювачів є короткий час старту робіт по верифікації та хороші можливості для налагодження.

Недолік полягає у високій вартості при надлишкових можливості для деяких завдань, таких, наприклад, як тестування контролера периферійного інтерфейсу. Частота моделювання істотно нижче у порівнянні зі спеціальними платами для налагодження ПЛІС-прототипів.

ПЛІС-прототипи.

Висока швидкість емуляції, в сотні МГц, досягається за допомогою модулів на основі ПЛІС. Можна використовувати як готові, так і спеціально розроблені налагоджувальні плати. Можливо безпосереднє вбудовування ПЛІС-прототипу в реальну систему, наприклад, шляхом установки в роз'єм PCI-e IBM-PC сумісної ЕОМ. Плата налагодження містить в собі різні пристрої: генератори тактових сигналів, приймачі Ethernet, високопродуктивне ОЗУ, відео інтерфейси тощо.

Існують три підходи до налагодження проектів на ПЛІС: використання вбудованого логічного аналізатора, використання зовнішнього контрольно-вимірювального обладнання та їх комбінування.

Вбудований логічний аналізатор передбачає розміщення на ПЛІС апаратних засобів налагодження, призначених для відстеження внутрішніх станів в моменти спрацьовування тригерів, що встановлюються розробниками. Зазвичай аналізатор вбудовується в ПЛІС за рахунок додавання одного або декількох IP-ядер. Для розміщення логіки вбудованого аналізатора і буферів пам'яті потрібно від 5% до 10% ресурсів ПЛІС. Через порт JTAG ядро аналізатора можна конфігурувати і динамічно керувати ним. Цей порт використовується і для виведення даних. Вбудований логічний аналізатор дозволяє відстежувати будь-які внутрішні сигнали в реальному часі. Для отримання доступу до сигналів робота системи зупиняється, ланцюжки сканування (scan chains) разом із захопленими даними передаються з ПЛІС. Потім отримана інформація аналізується на IBM-PC з використанням програм з графічним інтерфейсом або текстової консолі.

Налагодження проектів на ПЛІС може проводитися з використанням зовнішнього контрольно-вимірювального обладнання, такого як цифрові осцилографи змішаних сигналів або логічні аналізатори. Провідні виробники - компанії Tektronix і Agilent. Прикладами є осцилографи змішаних сигналів серії MSO4000 і логічні аналізатори серії TLA компанії Tektronix.

Технологічні плати з ПЛІС повинні мати спеціальні роз'єми для підключення зовнішніх пристроїв. Складність налагодження обумовлена обмеженим доступом до внутрішніх сигналів ПЛІС, взаємним впливом сигналів на друкованій платі. Багато пристроїв мають обмежене число виводів (зовнішніх контактів), менше, ніж кількість сигналів, необхідних для налагодження. Оскільки можливості вбудованих логічних аналізаторів обмежені, існують програми, такі як Altera Signal Probe та інші, які дозволяють вибирати потрібні внутрішні сигнали і перенаправляти їх на вільні контакти ПЛІС.

Можливе комбінування вбудованих ядер налагодження і зовнішнього контрольно-вимірювального обладнання. Наприклад, ядро Agilent Trace Core 2 (ATC-2) в складі програми Xilinx Chipscope Pro Analyzer спеціально розроблено для підключення зовнішнього логічного аналізатора Agilent і забезпечує доступ аналізатора до внутрішніх мереж проекту на ПЛІС. Логічний аналізатор забезпечує завдання складних умов захоплення даних. Для збереження даних використовується великий обсяг ОЗУ, що дозволяє відстежувати рознесені в часі події. Можливо порівняння сигналів проекту на ПЛІС з іншими сигналами системи

Зовнішнє контрольно-вимірювальне обладнання дорожче вбудованого логічного аналізатора, але має певні переваги. Цифровий осцилограф дозволяє запускатися від широкого спектра аналогових і цифрових сигналів, а також від сигналів послідовних шин, захоплювати їх з різною роздільною здатністю за часом. Зовнішній логічний аналізатор відкриває доступ до різних станів запуску і може захоплювати дуже довгі послідовності даних з високою роздільною здатністю за часом. Зовнішній осцилограф дозволяє

захоплювати до 10 млн точок, а логічний аналізатор до 256 млн. Є перевага в динамічності: не потрібно зупиняти роботу системи для аналізу захоплюваних даних. Зовнішнє обладнання дозволяє вивчати часові діаграми сигналів з роздільною здатністю менше наносекунди, тоді як вбудований логічний аналізатор може захоплювати дані тільки синхронно з внутрішньою тактовою частотою ПЛІС. Використання зовнішнього контрольно-вимірювального обладнання має деякі обмеження: отримання доступу до нового набору сигналів вимагає перекомпіляції проекту, що забирає час, може призвести до зміни часових характеристик і приховати шукані проблеми. Зазвичай число налагоджувальних висновків мало, що обмежує огляд і глибину аналізу.

Ці обмеження можна подолати шляхом комбінування налагоджувальних ядер і зовнішнього контрольно-вимірювального обладнання. Вибір конкретної методики залежить від особливостей проекту. Якщо налагодження обмежується функціональними проблемами всередині ПЛІС, то достатньо можливостей вбудованого логічного аналізатора. Зовнішнє обладнання краще при визначенні граничних значень часових характеристик, зіставленні внутрішньої активності ПЛІС з роботою інших пристроїв на друкованій платі.

Недолік налагодження проектів на ПЛІС полягає в відсутності метрик покриття RTL-коду і метрик покриття функціональності. Тести в основному знаходять помилки, пошук яких ведеться цілеспрямовано. Тому деякі помилки можуть залишитися непоміченими. Виявити їх допомагають випадкові тести. Вважається, що затвердження System Verilog Assertions (SVA) скорочують час тестування проекту на 50%. При їх використанні поліпшується наблюдаемость проекту, спрощуються пошук і виправлення недоліків.

Формальна верифікація використовується великими компаніями для відповідальних вузлів, наприклад, арифметичних алгоритмів процесорів,

сопроцессоров, кеш-пам'яті, графічних процесорів, де потрібна робота колективу фахівців.

Імітаційне моделювання широко поширене і зазвичай застосовується для налагодження окремих блоків мікросхем. Частота моделювання становить десятки герц, що обмежує сферу застосування. Сильними сторонами є наявність метрик покриття, автоматична генерація тестів, методології OVM / UVM.

Апаратні прискорювачі забезпечують високу швидкість, зазвичай кілька МГц. Вони дозволяють впровадити тестування проектів SoC на системному рівні. При цьому забезпечується повна видимість проекту. Недолік полягає в великих розмірах і високу вартість апаратного прискорювача. Основна сфера застосування - мікропроцесори, графічні процесори, мережеві застосунки. Плати для створення ПЛІС-прототипів мають найбільшу швидкістю в сотні МГц, але меншими налагоджувальними можливостями. Розміри дозволяють вбудовувати їх у реальну систему, а вартість порівняно невелика.

Можливий комбінований підхід, що включає, наприклад, імітаційне моделювання окремих блоків і використання ПЛІС-прототипів для перевірки проекту в цілому.

3 РЕАЛІЗАЦІЯ МЕТОДІВ ВЕРИФІКАЦІЇ В СИСТЕМАХ ЛОГІЧНОГО УПРАВЛІННЯ НА ПЛІС

3.1 Графові моделі автоматних пристроїв керування та їх верифікація

Реалізацію розроблених процедур верифікації пристроїв логічного керування розглянемо на прикладі пристрою керування теаорвенттиоятопом

Об’єктом проектування є система управління тепловентилятором. Об’єктом управління є тепловентилятор, алгоритм роботи якого наступний. Спочатку тепловентилятор знаходиться у режимі очікування. Після включення режиму нагріву (сигнал Onn) починається розгін вентилятора до необхідної швидкості протягом часу T_1. Після цього включається нагрівальний елемент та виробляється тепле повітря. Через проміжок часу T_2 відбувається опитування датчика температури та, якщо температура не досягла критичного рівня, продовжується нагрів. Якщо температура досягла критичного рівня $T_к$, тепловентилятор переходить в режим охолодження (працює тільки вентилятор) та відбувається опитування датчика температури через проміжок часу T_3. При цьому слід враховувати, що $T_3 >> T_2$, що обумовлено значно меншою швидкістю охолодження, ніж при нагріванні. Якщо температура знизилася, знов включається нагрівальний елемент та продовжує вироблятися тепле повітря. В такому режимі тепло вентилятор працює доти, поки включений режим нагріву.

При виключенні режиму нагріву у будь-який момент часу, тепловентилятор переходить у режим остаточного (повного) охолодження, працює тільки вентилятор протягом часу T_4, і після цього тепловнтилятор остаточно вимикається і переходить у режим очікування.

На рис.3.1 наведена структура системи автоматичного управління тепло вентилятором, де КА – керуючий автомат, ТВ – тепловентилятор, ДТ – аналогово-цифровий датчик температури.

При цьому слід враховувати наступне. У відповідності до принципів проектування пристроїв керування виконавчі пристрої, датчики, сенсори та перетворювачі відносяться до об’єкту управління, а при верифікації моделі та налагодженні прототипу керуючого автомату їх вихідні сигнали імітуються у алфавіті {0, 1}.

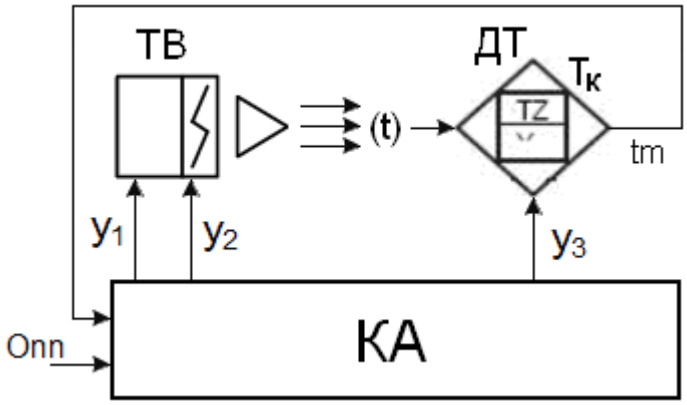

Рисунок 3.1 –Система автоматичного управління тепловентилятором

У відповідності до методики проектування автоматних систем логічного управління, викладені в підрозділі 1.1 цієї роботи, визначимо технічні стани роботи тепловентилятора.

1. Тепловентилятор виключений (режим очікування).
2. Режим включення живлення та розгону вентилятора.
3. Режим нагріву та подачі теплого повітря.
4. Режим охолодження нагрівального елементу.
5. Режим повного охолодження нагрівального елементу. (

Визначимо вхідні та вихідні сигнали для тепловентилятора, а також часові параметри його роботи:

Onn = {0, 1} – вхідний сигнал включення живлення тепловентилятора;

St = {0, 1} – вхідний сигнал включення режиму нагріву;

y_1 = {0, 1} – вихідний сигнал включення вентилятора;

y_2 = {0, 1} – вихідний сигнал включення нагрівального елементу;

$y_3 = \{0, 1\}$ – вихідний сигнал опитування датчика температури;

tm = {0, 1} – вихідний сигнал датчика температури, вхідний для керуючого автомата (tm = 1 при $t > T_к$, tm = 0 при $t \leq T_к$, де t – температура теплого повітря, $T_к$, – критична температура теплого повітря, вище якої нагрів не здійснюється);

T_1=3 сек – проміжок часу для розгону вентилятора до необхідної швидкості;

T_2=0,5 сек – проміжок часу між опитуваннями датчика температури при нагріві;

T_3=5 сек – проміжок часу між опитуваннями датчика температури при охолодженні;

T_4=20 сек – проміжок часу для повного охолодження нагрівального елементу.

Визначимо стани керуючого автомата. Виходячи з того, що всі технічні стани тепловентилятора є стійкими, аварійні стани не розглядаються, в розщепленні технічних станів немає потреби, кожному з технічних станів відповідає стан керуючого автомата $\{a_0, a_1, a_2, a_3, a_4\}$. Перелічимо стани керуючого автомата та відповідні вхідні та вихідні сигнали у кожному стані

1. Стан a_0 (тепловентилятор виключений), Onn=0, $y_1 = y_2 = y_3 = 0$.

2. Стан a_1 (режим включення та розгону вентилятора), Onn=1, tm=0. y_1=1, затримка T_1.

3. Стан a_2 (режим нагріву та подачі теплого повітря), Onn=1, $y_1 = y_2 = y_3 = 1$, затримка опитування T_2.

4. Стан a_3 (режим охолодження нагрівального елементу), Onn=1, $y_1 = y_3 = 1$, $y_2 = 0$, затримка T_3 .

5. Стан a_4 (режим повного охолодження нагрівального елементу), Onn=0, $y_2 = 1$, $y_1 = y_3 = 0$, затримка T_4 .

Для реалізації керуючого автомата будемо використовувати часову модель часового автомата Мура без вхідних обмежень та вихідних затримок, таймаути автомата Мура будуть реалізовані шляхом реалізації затримок у

станах автомата Мура. Граф переходів керуючого автомата (перша версія) наведений на рис3.2.

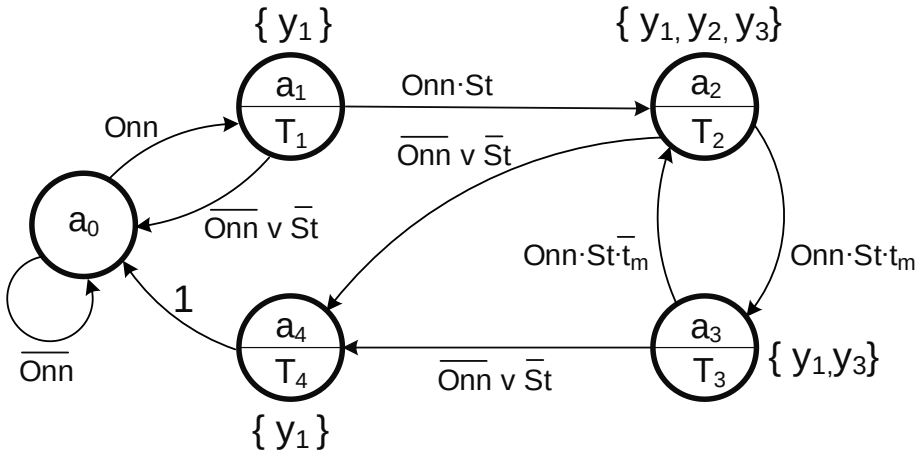

Рисунок 3.2 – Граф переходів КА тепло вентилятора (початкова версія)

Виконавши аналіз повноти переходів у кожному стані КА, у відповідності до методики, викладеної у підрозділі 2.1, можна зробити висновок, що системи переходів у станах a_2 та a_3 є неповними, тобто перехід $Onn \cdot St \cdot \overline{tm}$ у стані a_2 та перехід $Onn \cdot St \cdot tm$ у стані a3 невизначені. Для забезпечення повноти переходів у зазначених станах внесемо відповідні зміни у граф переходів (рис. 3.3).

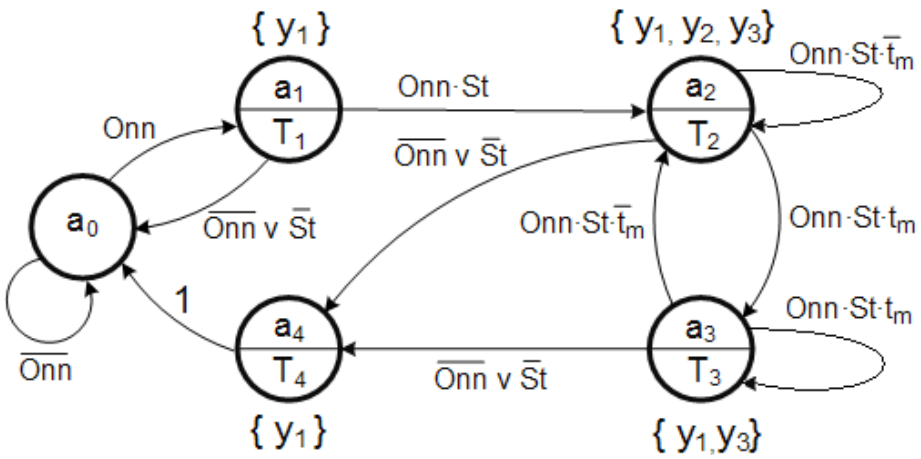

Рисунок 3.3 – Граф переходів КА тепло вентилятора (повна версія)

При проведенні подальшого аналізу графу переходів на надлишковість, можна зробити висновок, що сигнал St у графі є надлишковим, тобто сигнал включення живлення Onn може зразу включати і вентилятор, а також виключення живлення виключає режим подачі теплого повітря. Таким чином, остаточна версія графа переходів КА наведена на рис. 3.4.

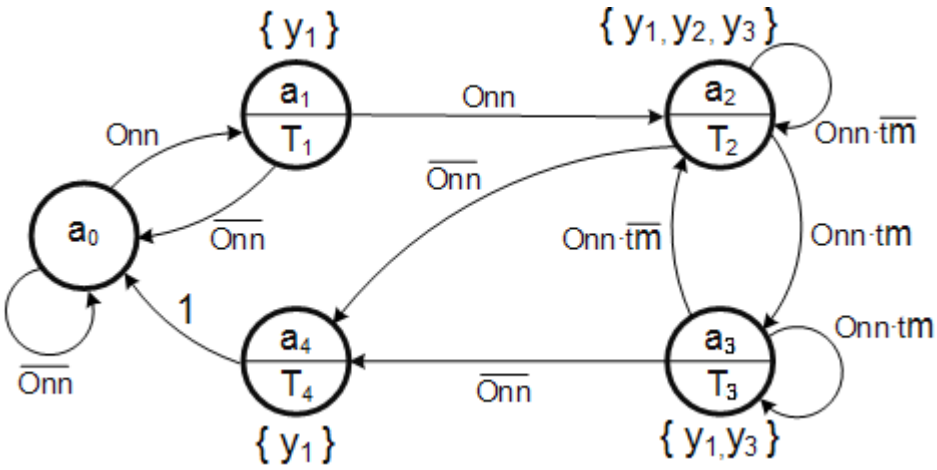

Рисунок 3.4 – Остаточна графова модель керуючого автомата

На основі наведеного на рис. 3.4 графа переходів КА може бути побудована HDL-модель керуючого автомата у стилі автоматного шаблона, фрагмент якої наведений на рис.3.5, а повна версія у додатку Б.

```
process (state, Onn, count, temp_treshold)
  begin
        next_count <= (others => '0');
        case state is
        . . .
        when a1 =>
                if count < T1 - 1 then  next_state <= state;
                        next_count <= count + 1;
                elsif Onn = '0' then  next_state <= a0;
                else    next_state <= a2;
                end if;
        . . .
        Y1 <= '1' when ( state /= a0 ) else '0';
        Y2 <= '1' when ( state = a2 ) else '0';
        Y3 <= '1' when (( state = a2 ) or (state = a3))    else '0';
```

Рисунок 3.5 – Фрагмент HDL-моделі керуючого автомата

3.2 Функціональна верифікація HDL-моделей автоматних пристроїв керування

При проведенні функціональної верифікації HDL-моделі керуючого автомата пристрою логічного керування будемо використовувати динамічну(тестову) верифікацію з використанням середовища TestBench системи моделювання Active-HDL. При проведенні функціональної верифікації HDL-моделей використовуються метрики покриття функціональних режимів, покриття блоків, покриття операцій, покриття асерцій, покриття фрагмепантів HDL-коду тощо.

У випадку наявності графової моделі Мура для автоматного пристрою керування функціональне покриття специфікації може бути зведено до обходу всіх станів автомата Мура та реалізації усіх переходів у кожному стані. При проведенні верифікації автоматних моделей реалізується так званий «неруйнівний» діагностичний експеримент (ДЕ) при проведенні якого тест (обхід графа) починається з початкової вершини графа переходів і повертається в цей же стан. По методиці, викладеній у [12], будується дерево рішень для побудови вказаного неруйнівного діагностичного експерименту (рис. 3,6). Такий ДЕ гарантує досяжність всіх станів керуючого автомата та реалізацію всіх переходів у кожному стані.

Особливістю наведеного дерева є присутність у ньому «неявного» циклу, який виникає при наявності на шляху обходу графа однойменних вершин, в нашому випадку a_2. Кількість циклів при верифікації в такому випадку визначає замовник.

Продемонструємо проведення ДЕ по обходу графа переходів керуючого автомата в основному режимі роботи. Перелічимо режими роботи тепловентилятора, які верифікуються : режим очікування (a_0) – режим розгону (a_1) – режим нагріву (a_2) – режим охолодження (a_3) – режим нагріву (a_2) – режим остаточного охолодження (a_4) – режим очікування(a_0).

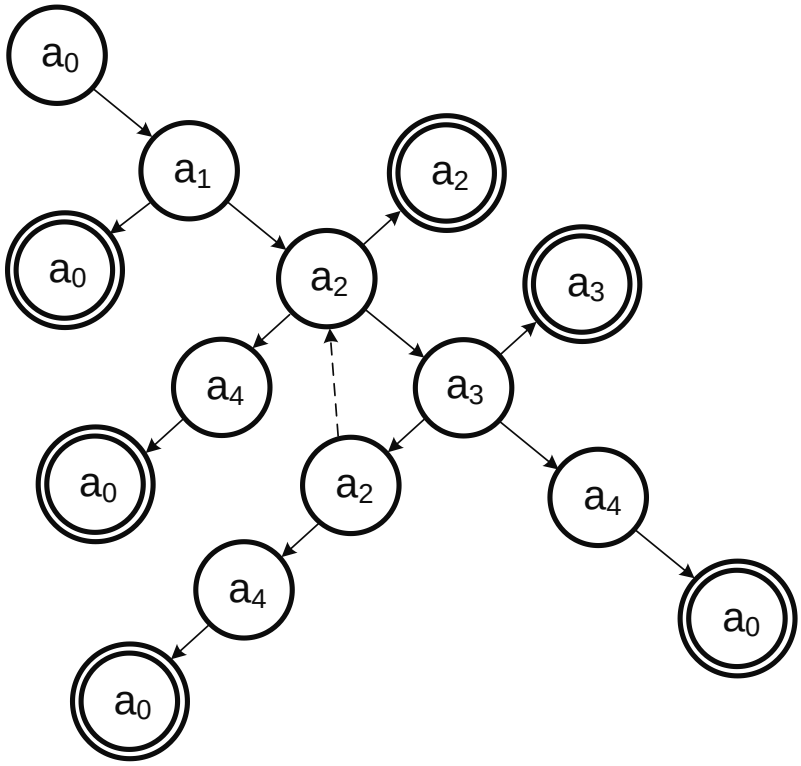

Рисунок 3.6 – Дерево рішень для побудови неруйнівного ДЕ

На рис.3.7 наведений фрагмент TestBench для наведеного варіанту обхода графа, а повна версія TestBench наведена у додатку В.

```
--      The process where the positive device scenario is viewed

        main_process: process
        begin

                -- Start position
                -- Reset all the inputs
                --
                Onn <= '0';
                Temp_treshold <= '0';

                -- Waiting for 2 tacts and then On the fan_heater
                -- (a0 -> a1)
                wait for period*2;
                Onn <= '1';

                -- The 5 tacts were elapsed. The fan speed is normal
                -- Then start to heat the fan heater
                -- (a1-> a2)
                wait for (a1_T-1) * period;
```

```
-- The 10 tacts were elapsed
-- Then start to heat the fan heater
-- (a2->a3)
wait for (a2_T) * period * 10;
Temp_treshold <= '1';
```

Рисунок 3.7 – Фрагмент TestBench для наведеного варіанту обходу графа

На рис.3.8 наведений Waveform для вказаного режиму функціональної верифікації HDL-моделі керуючого автомата системи управління тепло вентилятором.

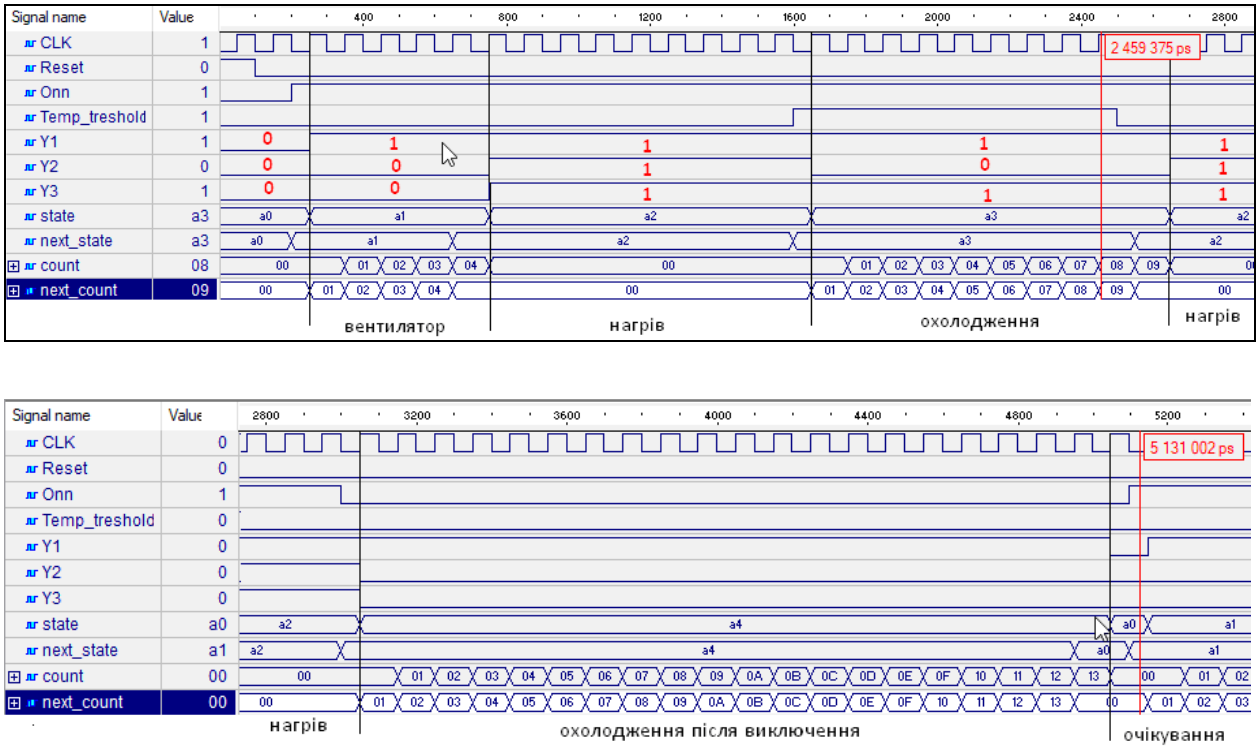

Рисунок 3.8 – Waveform для режиму функціональної верифікації при обході графа $a_0 – a_1 – a_2 – a_3 – a_2 – a_4 – a_0$.

Результати моделювання підтвердили функціональну відповідність розробленої HDL-моделі специфікації.

3.3 Апаратне тестування пристрою логічного керування на ПЛІС у реальному часі

Складність застосування традиційних методів налагодження для систем останнього покоління пов'язана з цілим рядом факторів. Найбільший вплив надає зростання складності сучасних проектів, розміщених всередині одиночної інтегральної схеми. У цій ситуації далеко не всі помилки проекту можуть бути виявлені і локалізовані при проведенні експериментів тільки над зовнішніми контактами. Основні труднощі налагодження породжуються необхідністю завершення експериментів до заданого часу.

Ситуація докорінно змінилася з появою ПЛІС і постійним зростанням числа систем, орієнтованих на їх застосування. При наявності достатніх вільних ресурсів у кристалі ПЛІС (внутрішньої логіки і зовнішніх контактів), а також можливості легкого підключення налагоджувальної апаратури досить універсальним і зручним рішенням виявляється підключення навіть найпростіших пристроїв введення / виводу. У подібних ситуаціях швидкість налагодження починає істотно залежати від досвіду і мистецтва розробника. Такі ж можливості (а іноді і більш складні) надають спеціальні налагоджувальні плати (демонстраційні, оціночні або прототипні) з установкою на них ПЛІС, відповідних за параметрами майбутнього проекту.

В роботі [13] запропоновано в якості тестера при налаштуванні проектів на ПЛІС використовувати зовнішній комп’ютер, поєднаний з платою налаштування через інтерфейс JTAG (рис. 3.9). Найбільш повну картину поведінки зовнішніх і внутрішніх сигналів системи , що налагоджується, можна отримати, розмістивши всередині кристалу логічний аналізатор, підключений до схеми, що проектується. Таке завдання легко вирішується при налагодженні проектів, які розміщені всередину кристалів ПЛІС, якщо після імплементації проекту в ПЛІС залишаються невикористані ресурси.

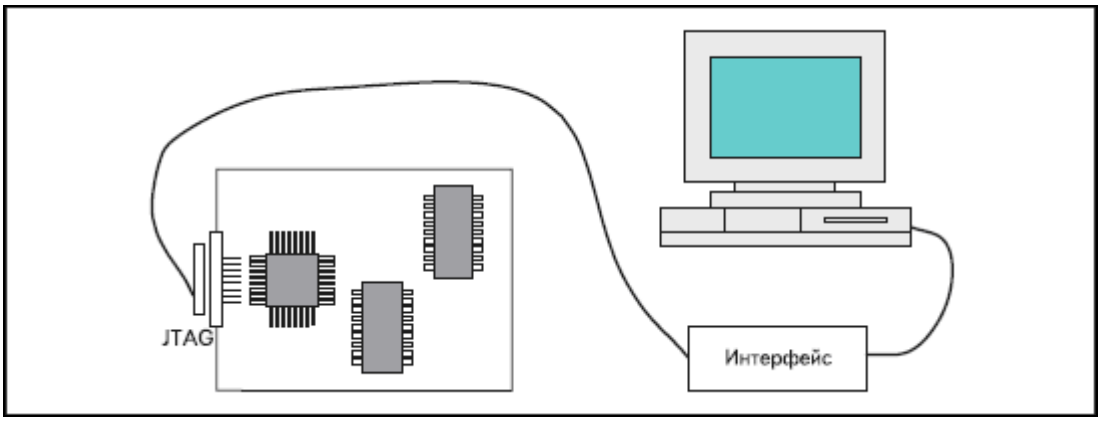

Рисунок 3.9 – Схема внутрикристального налаштування

Але такий підхід вимагає фактичної імітації системи Boundary Scan, що навряд доцільно для невеликих проектів. Тому запропоновано налагоджувальний тестер теж розміщувати в кристалі ПЛІС при наявності вільних ресурсів. Для проекту пристрою керування тепловентилятором запропоновано поєднання КА з тестером, у вигляді структурної моделі на одному кристалі (по аналогії з TestBench при функціональній верифікації) (рис.3.10).

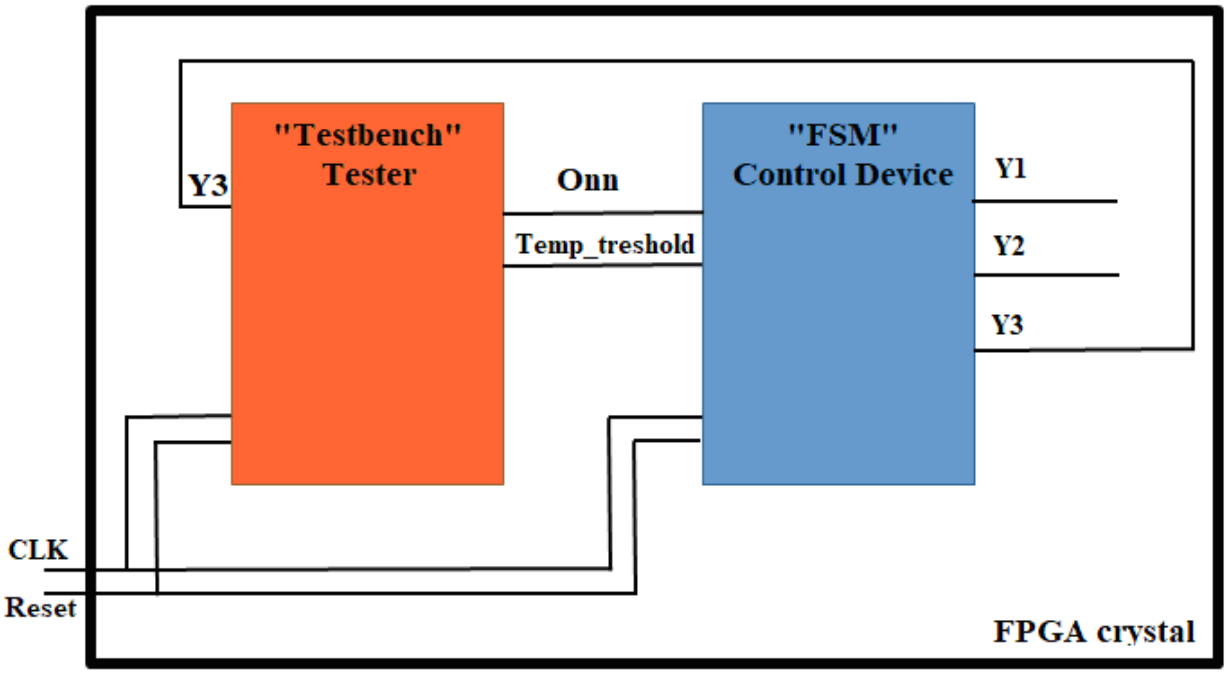

Рисунок 3.10 – Структура тестера і керуючого пристрою на одному кристалі

Тестер має 2 вихідних сигнали Onn і Temp_treshold, які є входами для КА. У свою чергу вихід КА y3 є входом для тестера, високий рівень якого ініціює «імітацію» роботи датчика температури.

Затримки можна розглядати у метричному часі (у вигляді секунд) або у автоматному часі (у вигляді тактів). Передбачається, що макетування пристрою буде виконуватися при частоті в 1 Гц, що в свою чергу дає період в 1с. У разі даного способу моделювання, з метою скорочення його часу, затримки вимірюються в тактах.

При моделюванні схемного прототипу часового автомата у єдиному кристалі з тестером використовувалися такі затримки.

Таймаути для проекта: $T_1 = 5$, $T_2 = 1$, T3 = 10, T4 = 20

Таймаути для тестера: T5 = 2, T6 = 31, T7 = 10.

Моделювання і синтез здійснювалося інструментальними засобами САПР Xilinx ISE.

Початок моделювання відбувається за стандартним сценарієм. Виконується скидання сигналу Reset на задньому фронті 1 такту. На другому такті з'являється сигнал високого рівня Onn, що означає перехід зі стану зі стану a_0 в стан a_1. Слід зауважити, що для управління сигналом Onn створений окремий лічильник з сигналами onn_count і next_onn_count.

Відрахування часу натискання кнопки починається з моменту появи високого рівня сигналу до появи низького рівня. Тривалість часу натискання розрахована на 30 тактів, демонструючи повний цикл роботи тепловентілятора за позитивним сценарієм (обхід всіх станів).

Лістинг 3.1 – Процес натискання / віджимання кнопки

```
process (onn_count)
      begin
            if onn_count < T5-1 then Onn <= '0';
                  next_onn_count <= onn_count + 1;
            elsif onn_count >= T5-1 and onn_count <T6-1 then Onn <= '1';
                  next_onn_count <= onn_count + 1;
            else  Onn <= '0'; next_onn_count <= (others => '0');
            end if;
      end process;
```

Лістинг 3.2 – Список затримок для тестера

```
-- delays for testbench
        --
        -- delay for Onn <= '1'
        --
        constant T5 : std_logic_vector ( count_length - 1 downto 0 ) := "00010";          -- 2

        -- all the testbench time working (delay for Onn <= '0')
        --
        constant T6 : std_logic_vector ( count_length - 1 downto 0 ) := "11111";          -- 31

        -- delay for heating/cooling
        --
        constant T7 : std_logic_vector ( count_length - 1 downto 0 ) := "01010";          -- 10
```

На рис.3.11 наведено фрагмент часової діаграми, який відповідає початку роботи тепловентилятора.

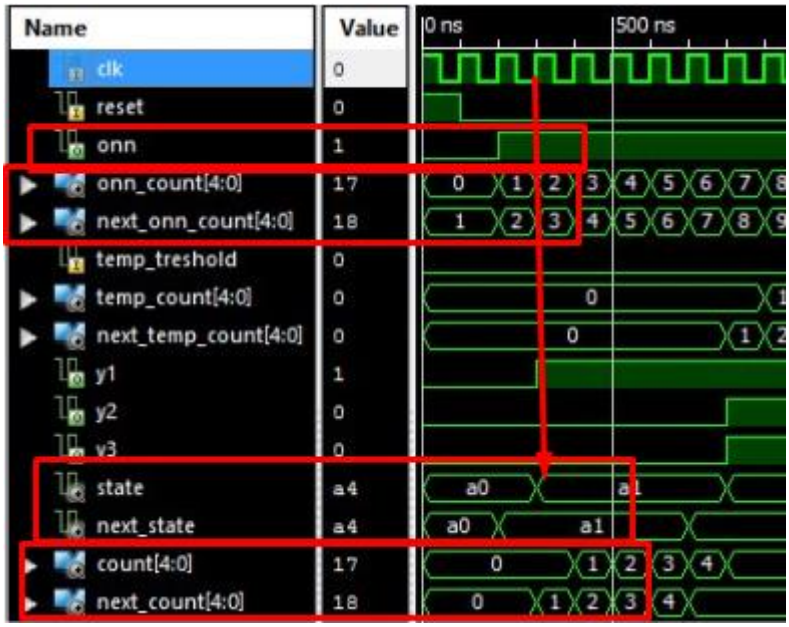

Рисунок 3.11 – Перехід зі стану a_0 у стан a_1

Перебуваючи в стані a1, КА, в свою чергу ініціює 5-ти тактову затримку стану. В цей час вентилятор набирає обертів. Після закінчення часу відбувається перехід зі стану a_1 в стан a_2. Якщо ж сигнал Onn знаходиться в низькому рівні, то відбувається перехід в стан a_0.

Слід зауважити, що, як і в випадку з сигналом Onn і з тестером в цілому, HDL-модель УУ також має лічильник з сигналами count та next_count, який відповідає за прийняття часу затримки кожного стану по високому рівню синхросигналу і відліку часу до переходу зі стану в стан.

Лістинг 3.3 – Фрагмент опису кінцевого автомата в стані a1

```
when a1 =>
                        if count < T1 - 1 then
                              next_state <= state;
                              next_count <= count + 1;
                        elsif Onn = '0' then next_state <= a0;
                        else  next_state <= a2;
                        end if;
```

Лістинг 3.4 – Список затримок для КА

```
-- delays for every state
--
constant T1 : std_logic_vector ( count_length - 1 downto 0 ) := "00101";          -- 5
constant T2 : std_logic_vector ( count_length - 1 downto 0 ) := "00001";          -- 1
constant T3 : std_logic_vector ( count_length - 1 downto 0 ) := "01010";         -- 10
constant T4 : std_logic_vector ( count_length - 1 downto 0 ) := "10100";          --20
```

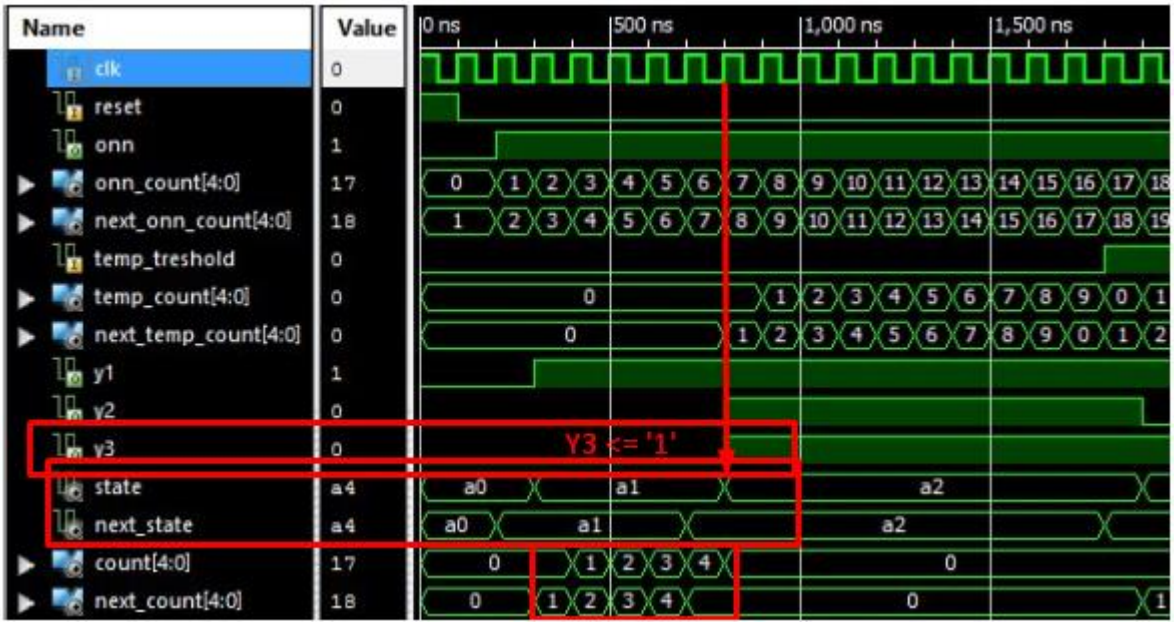

Рисунок 3.12 – Перехід зі стану a_1 в стан a_2 з демонстрацією затримки стану

У стані a_2 відбувається включення основної функції тестера: імітація вимірювання температури нагрівача і передача УУ сигналу про перевищення максимально допустимої температури (Temp_treshold). Для спрощення процесу моделювання було прийнято умова зміни сигналу Temp_treshold кожні 10 тактів, що свідчить про створення окремого таймера для даного процесу з сигналами temp_count і next_temp_count.

Виконання дій в даному процесі ініціюється передачею високого рівня сигналу Y3 від КА на вхід тестеру.

Лістинг 3.5 – Процес, що імітує роботу і опитування датчика температури

```
process (Y3,temp_count)
     begin
          Temp_treshold <= temp_sensor;
          next_temp_count <= (others => '0');

          if(Y3 = '1') then
               if temp_count < T7-1 then
                  next_temp_count <= temp_count + 1;
               else
                  temp_sensor <= not temp_sensor;
               end if;
          else
               temp_sensor <= '0';
          end if;
     end process;
```

У свою чергу, КА кожен такт перевіряє рівень цього сигналу. При відсутності високого рівня автомат залишається в стані a_2. При наявності високого рівня сигналу на вході відбувається перехід в стан a_3. Тим самим відбувається підвищення температури нагрівача.

Також в разі низького сигналу Onn, поданого на вхід УУ, відбувається перехід зі стану a_2 в стан a_4.

Лістинг 3.6 – Фрагмент опису кінцевого автомата в стані a_2

```
when a2 =>
      if count < T2 - 1 then
            next_state <= state;
            next_count <= count + 1;
      elsif Onn = '0' then       next_state <= a4;
      elsif (Onn = '1' and Temp_treshold = '1') then
            next_state <= a3;
      else       next_state <= a2;
      end if;
```

Моделювання переходу зі стану a2 в стан a3 триває 10 тактів.

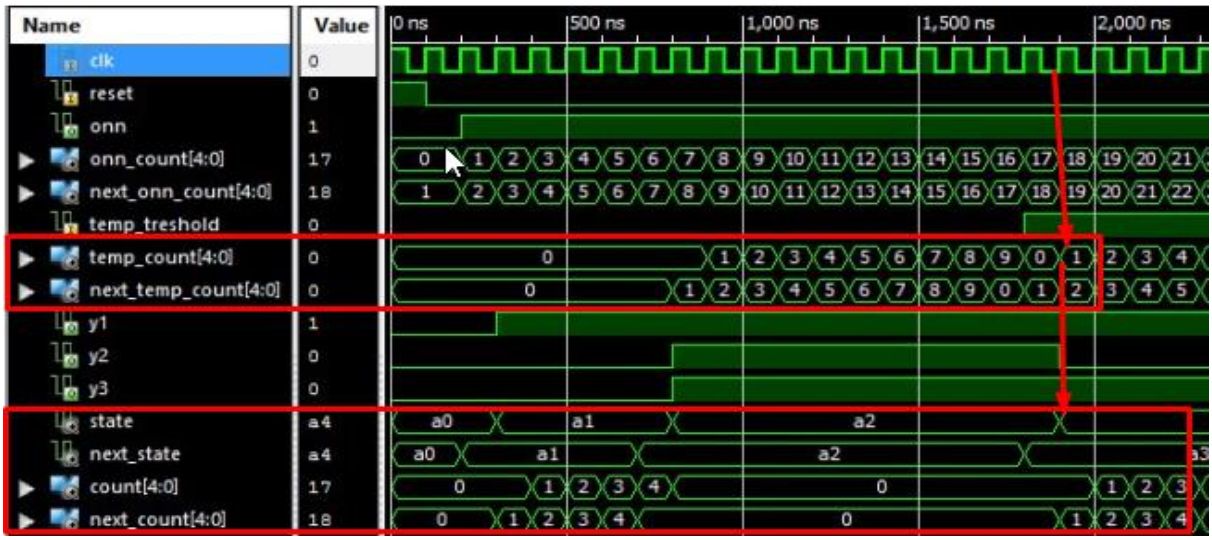

Рисунок 3.13 – Перехід зі стану a2 до стану a3

При досягненні температури, яка дорівнює критичній або вище, відбувається відключення нагрівального елементу і деякий час він остуджують. Цій дії відповідає стан a_3, в яке відбувається перехід при високому рівні сигналу Temp_treshold. Даний стан має таймаут в 10 тактів. Після чого відбувається перевірка того, що на вході сигнал Temp_treshold має низький рівень потім здійснюється перехід з поточного стану в стан a_2. В іншому випадку автомат знаходиться в цьому ж стані, таймер знову починає відлік 10 тактів.

Однак необхідно зауважити, що, як і в стані a_2 відбувається перевірка високого рівня сигналу Onn. При наявності високого рівня сигналу на вході автомат виконує вищевказані дії. Інакше відбувається перехід в стан a_4.

Лістинг 3.7 – Фрагмент опису кінцевого автомата в стані a_3

```
when a3 =>
                if count < T3 - 1 then
                        next_state <= state;
                        next_count <= count + 1;
                elsif Onn = '0' then
                        next_state <= a4;
                elsif (Onn = '1' and Temp_treshold = '0') then
                        next_state <= a2;
                else    next_state <= a3;
                end if;
```

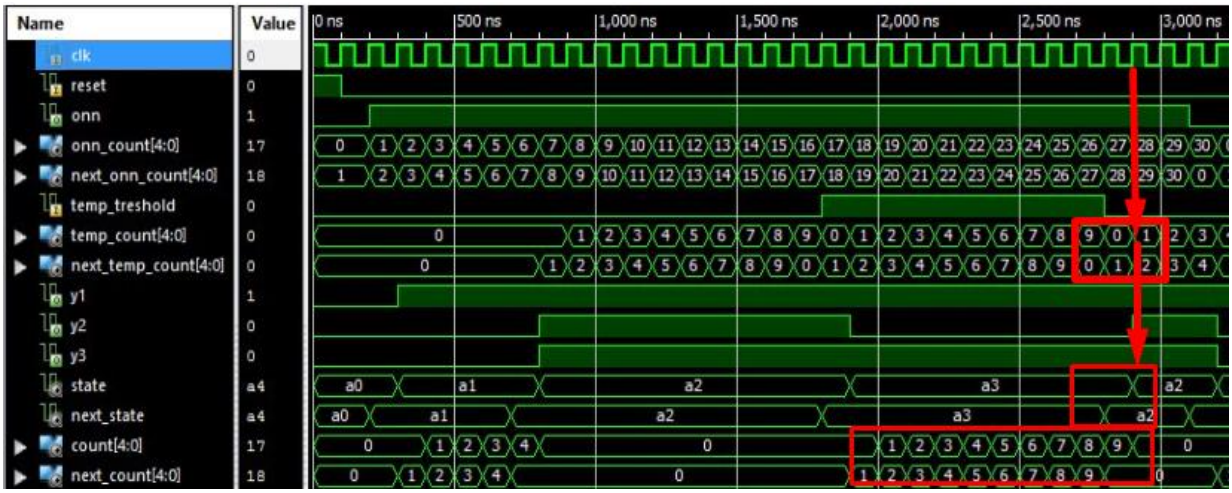

Рисунок 3.14 – Перехід зі стану a_3 до стану a_2

У разі низького рівня сигналу Onn, в будь-якому з станів, крім a0, здійснюється перехід в стан a4, що сигналізує про відключення пристрою через певний проміжок часу (20 тактів), коли нагрівальний елемент буде охолоджуватися. Після закінчення часу очікування відбувається безумовний перехід в стан a_0 (повернення в початковий стан) і пристрій знову почне свою роботи при високому рівні сигналу Onn.

У цьому стані робота тестера по вимірюванню температури припиняється, а симуляція високого рівня сигналу Onn повторно починається, щоб продемонструвати те, що реакція на даний сигнал в цьому стані відсутня.

Листинг 3. 8 – Фрагмент опису кінцевого автомата в стані a_4

```
when a4 =>
      if count < T4 - 1 then
            next_state <= state;
            next_count <= count + 1;
      else  next_state <= a0;
      end if;
```

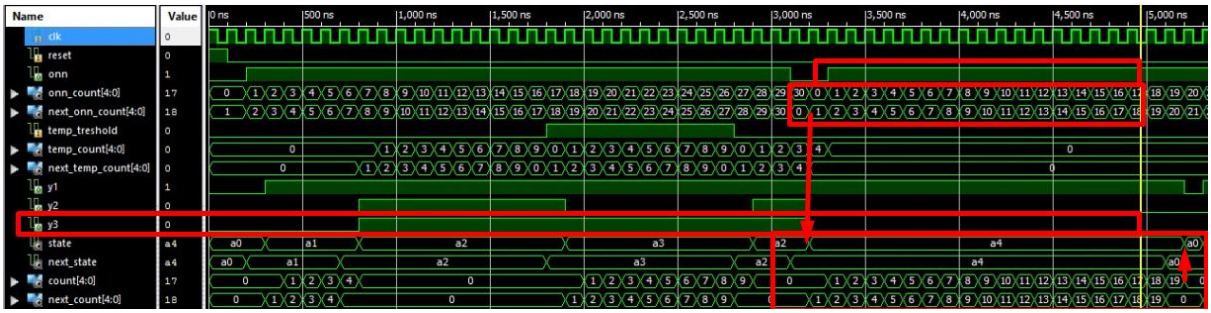

Рисунок 3.15 – Перехід до стану a_4

Повна HDL-модель керуючого автомату з тестером наведена у додатку Г.

Результати синтезу для ПЛІС XC3S500E (ПЛИС FPGA Xilinx сімейства Spartan-3E, 10476 еквівалентних логічних елементів) наведені на рис.3.16.

Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <U1/state/FSM> on signal <state[1:3]> with user encoding.
State | Encoding a0 | 000 a1 | 001 a2 | 010 a3 | 011 a4 | 100
Registers 17 Flip-Flops 17 Cell Usage :# BELS 59

Рисунок 3.15 – Результати синтезу структурної моделі

Засинтезовано 17 тригерів: 3 тригера для 5 станів; 4 тригера для лічильника temp_count; 5 тригерів для лічильника onn_count; 5 тригерів для лічильника count. При синтезі на XC3S500E по числу комбінаційних елементів результати використання двох процесів (next_count) (59 BELS).

ВИСНОВКИ

Розглянуто роль та місце верифікації на різних етапах автоматизованого проектування пристроїв керування в системах логічного управління. В якості пристрою керування розглядається часовий автомат Мура, моделлю його є темпоральній граф переходів. Для автоматизованого проектування графова модель керуючого автомата описується на мові опису апаратури VHDL у формі автоматного шаблону.

Проведений аналіз коректної побудови функцій умов переходів графових моделей керуючих автоматів, представлених у вигляді HDL-моделей синтезованої підмножини, с точки зору подальшого автоматизованого синтезу. Показано, що проблеми, пов'язані з некоректними умовами, можуть проявитися тільки в процесі синтезу, тому перевірку умов переходів на несуперечність і повноту необхідно виконувати ще на етапі формування моделі графу переходів автомата

Формальна верифікація використовується для відповідальних вузлів, наприклад, арифметичних алгоритмів процесорів, сопроцессоров, кеш-пам'яті, графічних процесорів, де потрібна робота колективу фахівців.

Імітаційне моделювання широко поширене і зазвичай застосовується для налагодження окремих блоків мікросхем. Частота моделювання становить десятки герц, що обмежує сферу застосування. Сильними сторонами є наявність метрик покриття, автоматична генерація тестів, методології OVM / UVM.

Апаратні прискорювачі забезпечують високу швидкість, зазвичай кілька МГц. Вони дозволяють впровадити тестування проектів SoC на системному рівні. При цьому забезпечується повна видимість проекту. Недолік полягає в великих розмірах і високу вартість апаратного прискорювача. Основна сфера застосування - мікропроцесори, графічні процесори, мережеві застосунки. Плати для створення ПЛІС-прототипів

мають найбільшу швидкістю в сотні МГц, але меншими налагоджувальними можливостями. Розміри дозволяють вбудовувати їх у реальну систему, а вартість порівняно невелика.

Розроблені методи апробовані на моделі керуючого автомата в системі управління тепловентилятором. Моделювання виконувалося в середовище Activ-HDL, синтез та імплементація схеми здійснена на платі Spartan 3E інструментальними засобами Xilinx ISE.

Наукова новизна роботи полягає у подальшому розвитку методів програмно-арпаратної верифікацій моделей автоматних пристроїв керування на різних рівнях автоматизованого проектування (алгоритмічному, функціонально-блочному, схемотехнічному та конструкторсько-технологічному).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления / А.А. Шалыто. – СПб.: Наука, 1998. – 628 с.

2. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации. – СПб.: Наука, 2000. – 780 с.

3. Шалыто А. А. Автоматное программирование / Н.И. Поликарпова, А.А. Шалыто. – Спб.: Питер, 2008. – 168 с.

4. Harel D. Statechars: a Visual Formalism for complex systems / D. Harel // Science of Computer Programming. – 1987. – Vol. 8. – P. 231-274.

5. Alur R. A theory of timed automata / R. Alur, D.L. Dill // Theoretical Computer Science. – 1994. – V.126. – N 2. – P. 183-235.

6. Haskell R. Digital Design Using Digilent FPGA Boards - VHDL / Active-HDL Edition / Richard E. Haskell, Darrin M. Hanna. – LBE Books Rochester Hills, MI, 2009. – 381 p.

7. Норенков И.П. Основы автоматизированного проектирования: учеб. для вузов / И.П.Норенков. – 4-е изд., перераб. и доп. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 430 с

8. Rashinkar P. System-on-a-chip Verification: Methodology and Techniques Cadence Design Systems / P. Rashinkar, P. Paterson, L. Singh. – Kluwer Academic Publishers, 2002. – 372 p.

9. Lam W. K. Hardware Design Verification: Simulation and Formal Method-Based Approaches. – Prentice Hall PTR, 2005. – 624 p.

10. Шкіль О.С., Аналіз коректності графів переходів керуючих автоматів при побудові HDL-моделей для автоматизованого синтезу / О.С Шкіль., Е.М. Кулак., І.В. Філіппенко, О.Г. Шакура, В.В. Фоменко // Радіоелектроніка та інформатика – 2019. – №1. – С. 28-34.

11. Слинкин Д.И., Анализ современных методов тестирования и верификации проектов сверхбольших интегральных схем / Д.И. Слинкин //

Программные продукты и системы (Software & Systems). – 2017. – Т.30. – № 3. – С.401-408.

12. Шкиль А.С. Поиск ошибок проектирования в HDL-моделях цифровых автоматов / С. Альмадхоун, Е.Е. Сыревич, А.С. Шкиль // Вестник Херсонского государственного технического университета.– 2013. – №2 (46). – С. 377-383

13. Грушвицкий Р. Проектирование в условиях временных ограничений: отладка проектов (часть 1) / Р. Грушвицкий, М. Михайлов // Компоненты и технологии. – 2007. – №6. – С. 131 – 136.