Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Подієві HDL-моделі керуючих автоматів в системах логічного управління
(тема)

Виконав: студент 2 курсу, групи СКСм-19-2

_____Малишенко Д.О._____
(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)

2021 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)

«__01__»___02____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Малишенко Дмитру Олександровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Подієві HDL-моделі керуючих автоматів в системах логічного управління_____-
_____
_____

Event HDL-models of Control Finite State Machines in Logic Control Systems__
_____

затверджена наказом по університету від «_26_»__03__2021 р. № _385 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____25.05.2021_____

3. Вихідні дані до роботи (проекту)_____
Мова опису апаратури VHDL_ _____
САПР XILINX ISE_ _____
ПЛІС Spartan 3E_____
_____

4. Перелік питань, що потрібно опрацювати у роботі _____
Подієвий стиль побудови програмних кодів_____
Методи проектування систем логічногго управління_____
Методи автоматного програмування при побудові моделейі автоматів_____
Класифікація HDL-моделей керуючих автоматів_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _28 слайдів_ _____ _____

_____

_____

_____

_____

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 01.02.2021_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	01.02.2021 -20.02.2021	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	20.02.2021 -10.03.2021	
3	Розробка принципів проектування автоматних систем логічного управління	10.03.2021 -31.03.2021	
4	Розробка принципів класифікації моделей керуючих автоматів	01.04.2021 -15.04.2021	
5	Розробка процедур побудови моделей подієвих автоматів Мура	15.04.2021 - 01.05.2021	
6	Розробка процедур моделей подієвих автоматів Мілі	01.05.2021 - 10.05.2021	
7	Моделювання, синтез та імплементація розроблених моделей	10.05.2021-20.05.2021	
8	Оформлення пояснювальної записки	20.05.2021-25.05.2021	
9	Захист проекту	25.05.2021-30.05.2021	

Студент _____ _____
(підпис)

Керівник роботи (проекту) ___ _____ _____доц. Шкіль О.С.____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 78 сторінок, 21 рисунок, 2 таблиці та 9 джерел за переліком посилань.

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

Метою роботи є розробка процедур побудови HDL-моделей керуючих автоматів в системах логічного управління та їх реалізацію на технологічній платформі ПЛІС з використанням мов опису апаратури.

Проведена класифікація моделей керуючих автоматів в системах логічного управління. За способом формування вихідних сигналів керуючі автомати класифікуються на моделі Мура і Милі, а за способом обробки вхідних сигналів на активні і пасивні. Визначено поняття події, як зміни зовнішнього середовища та наведена класифікація подій на ініціюючі та перекриваючі. Визначено поняття змішаного автомата, як такого у якого частина вхідних сигналів розглядається як вхідні дії (які опитуються), а частина вхідних сигналів розглядається як події.

Моделі часових автоматів представлені на мові опису апаратури VHDL в формі автоматного шаблону. Виконано поведінковий моделювання запропонованих моделей, синтез і імплементація в FPGA, а також моделювання після імплементації з використанням САПР Xilinx ISE 14.7.

ABSTRACT

The explanatory note contains: 78 pages, 21 figures, 2 tables, 9 sources according to the list of links.

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

The purpose of the work is to develop procedures for constructing HDL-models of finite state machines in logical control systems and their implementation on the FPGA technological platform using hardware description languages.

The classification of models of control finite state machines in logical control systems is carried out. According to the method of generating output signals, control finite state machines are classified into the Moore and Mealy models, and according to the method of processing input signals into active and passive ones. The concept of an event is defined as a change in the external environment and the classification of events into initiating and interrupting is given. The concept of a mixed finite state machines is defined as one in which part of the input signals is considered as input actions (which are interrogated), and part of the input signals is considered as events.

Timed finite state machines models are presented in the hardware description language VHDL in the form of an automaton patterns. Behavioural simulation of the proposed models, synthesis and implementation in FPGA, and simulation after implementation using CAD Xilinx ISE 14.7 were performed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГСА – граф-схеми алгоритмів;

ДП – дискретний пристрій;

ДС – дискретна система;

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;

CAD – Computer-Aided Design system (САПР);

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);

ISE – Integrated Synthesis Environment (інтегрована система синтезу);

RTL – Register Transfer Level (рівень регістрових передач);

SoC – System of Chip (система на кристалі);

UML – Unified Modeling Language (уніфікована мова моделювання);

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

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

ВСТУП

На основі загальної концепції побудови систем автоматизованого управління в них завжди модно виділити пристрої керування і керовані об'єкти. Дотримуючись цієї концепції, системи управління на основі кінцевих автоматів (Finite State Machines, FSM) також можна розділити на дві частини керуючу частину, що відповідає за реалізацію логіки алгоритму управління, тобто за вибір виконуваних дій, що залежить від поточного стану і вхідних сигналів, а також за перехід у новий технічний стан, та керовану частину, що відповідає за виконання функцій, обраних для виконання керуючої частиною, та за формування компонентів вхідних сигналів зворотних зв'язків для керуючої частини. Оскільки для реалізації керуючої частини в таких пристроях, як правило, використовуються кінцеві автомати, то вони мають назву керуючі автомати. Подібні пристрої широко застосовуються в системах логічного управління та системах Internet of Things.

Серед усієї множини систем управління значну частину складають системи логічного управління (СЛУ, Logical Control System), у яких керуючі сигнали приймають значення логічного нуля або одиниці в залежності від попадання значень фізичних величин параметрів сигналів від керованої частини у інтервали, граничних значень які визначені для цих параметрів. Для технічної реалізації керуючої частини зазначених систем найбільш придатною є модель структурного кінцевого автомата, а візуальним представленням алгоритму функціонування є граф переходів (State Diagram). Особливістю керуючих автоматів в системах логічного управління є наявність серед вхідних сигналів (Input Values) не тільки логічних сигналів від об'єкта управління, а й зовнішніх по відношенню до системі управління сигналів (подій) зовнішнього світу (External Events). Такі сигнали, як правило, забезпечують взаємодію автоматних систем логічного управління з

зовнішньої середовищем.

Більшість реальних систем логічного управління є системами реального часу і для їх реалізації прийнято використовувати модель часового автомата (timed FSM) а для візуального представлення темпоральний граф переходів (Temporal State Diagram). При автоматизованому синтезі часових автоматів з використанням мов опису апаратури для коректного синтезу схеми автомата з урахуванням часових параметрів, як правило, використовуються шаблони автоматного програмування на мовах опису апаратури (HDL). Для мови VHDL це спеціальна структура VHDL-моделі, в якій функції переходів і виходів виділені в окремі процеси (процес), призначення нового стану здійснюється в спеціальному процесі, пов'язаному з синхронізацією, а затримки реалізуються через лічильник автоматних тактів. Крім того, при обробці зовнішніх подій в системах реального часу необхідно враховувати часовий проміжок, протягом якого зовнішні подія може змінити алгоритм роботи керуючого автомата.

Метою даної роботи є розробка єдиного шаблону на мові VHDL для опису різних типів часових керуючих автоматів в системах логічного управління реального часу в стилі автоматного програмування, які реалізуються на апаратній платформі Field-programmable Gate Array (FPGA). Таким чином, розробка методів проектування моделей цифрових пристроїв на різних стадіях циклу проектування є актуальною та технічно значущою задачею.

Об’єкт дослідження у роботі – процедури автоматизованого проектування автоматних пристроїв логічного керування на технологічній платформі програмованих логічних інтегральних схем (ПЛІС).

Предмет дослідження – моделі, методи та процедури побудови шаблонів автоматних моделей пристроїв логічного керування на мовах опису апаратури, які реалізуються в ПЛІС FPGA

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз існуючих методик та стилів побудови програмних

кодів в системах автоматизованого проектування;

– провести подальший розвиток процедур побудови автоматних програм в системах логічного управління з використанням мов опису апаратури;

– розробити класифікацію автоматних моделей в залежності від способу обробки вхідних сигналів та методів формування вихідних сигналів;

– розробити процедури побудови шаблонів на синтезованій підмножині мови опису апаратури VHDL для проектів керуючих автоматів на технологічній платформі ПЛІС;

– реалізувати розроблені методики проектування автоматних моделей інструментальними засобами системи автоматизованого проектування (САПР) XILINX ISE з реалізацією в FPGA Spartan 3E.

1 СТИЛІ ТА МЕТОДИ ПОБУДОВИ ПРОГРАМНОГО КОДУ

1.1 Загальний огляд стилів побудови програмного коду

Інформатика накопичила величезний багаж експериментальних і теоретичних даних, які потребують серйозного осмислення. В інформатиці, оскільки вона займається найскладнішими розумовими конструкціями і є одним з небагатьох місць в практиці, де використовуються дійсно штучні об'єкти, багато проблем представлені в найбільш яскравому і тому вдячному для аналізу вигляді. Розвиток інформатики неможливо без з'єднання високої теорії і високою практики, що також робить її одним з найсерйозніших і найважливіших об'єктів аналізу. Накопичений багаж виключно сильних і тонких практичних і теоретичних методів (досить згадати функціональне програмування і топологічну теорію областей, абстрактні типи даних і алгебраїчну теорію уявлень) показує, що наука і практика підійшли до рівня зрілості. Більш того, інформатики досягла такої точки, коли просто не може розвиватися далі без системи знань і концепцій [1].

Підставами для виробленого аналізу є основні висновки логічної теорії та методології XX століття. Першою підставою служить принцип: жодна формалізація не є універсальною. Кожна формалізація обмежена, і, більш того, при досить серйозному аналізі сама підказує власні альтернативи.

Він є наслідком теорем Геделя про неповноту і теореми Тарського про невимовності істини. Таким чином, необхідно перш за все чітко знати межі застосування кожного методу і відмовитися від постановки по-проса про якомусь єдино правильному методі, яким можна вирішити всі завдання.

Уже конструктивні логіки для різних класів задач протирічать один одному в самих своїх підставах.

У нильпотентній логіці, яка описує побудову програм на базі глобальних незворотних процесів і локальних умов, приймається правило

виключеного застою:

У реверсивної логіці, яка описує побудову програм на базі глобальних оборотних дій і глобальних умов (наприклад, програм для квантового або надпровідного комп'ютера), виконана формула

Навіть якщо дві формалізації складного природного поняття формально не суперечать один одному, вони почнуть суперечити заважати при спробах глибшого аналізу або розвитку системи (т.зв. концептуальні суперечності). Навіть явні протиріччя наполегливо ігноруються і теорією, і практикою, зацикленої на позитивне мислення, а концептуальні взагалі не приймаються до уваги.

Тому з'єднувати всі мислимі кошти в одному і тому ж місці - вірний шлях до системам, що само розвиваються. Зауважимо, що таке з'єднання є загальним місцем в сучасній інформатиці, оскільки прийнято критикувати системи за те, чого в них немає. Виникаючі при додаванні нових можливостей концептуальні протиріччя нікого не хвилюють, поки не стає занадто пізно.

Другою підставою служить гильбертовськая концепція ідеальних і реальних об'єктів і парадокс винахідника. Більшість об'єктів і понять в областях, які мають розвиненим математичним апаратом, не мають прямих інтерпретацій в реальності. Але їх усунення, навіть якщо воно в деяких випадках теоретично можливо, призводить до неприйнятного (як мінімум, вежа експонент) розростання довжини доказів. Таким чином, манівці є найбільш короткими і ефективними.

Третьою підставою є система оцінки складності математичних понять по їх типам. Перший тип - об'єкти, другий тип - їх множина або функції над об'єктами, третій тип - функції над об'єктами другого типу тощо.

Система знань тим більше необхідна тому, що інформатика нерозривно пов'язана із сучасною економікою, в якій інду-стрия обману і самообману (реклама) займає вирішальне місце, то, що вплив цього духа проникло в сучасну науку у вигляді системи грантів, яка вимагає нестримної

самореклами заважає дійсно серйозного теоретичного і методологічного осмислення проблем.

Під стилем програмування розуміється загальна і внутрішня узгоджена сукупність базових конструкцій програм і способів їх композиції, що володіє загальними фундаментальними особливостями, як логічними, так і алгоритмічними. Стиль включає також сукупність базових концепцій, пов'язаних з цим-ми програмами.

Стиль програмування реалізується через методології програмування. У більшості випадків методологія полягає в сукупності угод про те, які базові концепції мов програмування і які їх поєднання вважаються прийнятними або неприйнятними для даного стилю. Методологія включає в себе, зокрема, модель обчислювача для даного стилю.

Методологія реалізується через методики, які складаються з таких компонентів:

– заохочення (або прямий припис) використання деяких базових концепцій програмування;

– заборона (або обмеження) застосування деяких інших базових концепцій, іноді заборона чи може бути неявним, через виключення небажаних концепцій з обраної мови або його діалекту;

– вимоги і рекомендації з оформлення і документування програм;

– сукупність інструментальних і організаційних засобів, що підтримують всі перераховані вище вимоги і рекомендації.

Стилі програмування природно класифікуються за трьома координатами: першорівневі і високорівневі, глобальність або локальність дій і умов.

Високорівневі стилі відрізняються тим, що функції і перетворення можуть служити повноправними значеннями.

Згідно з наведеними характеристиками, природно виділити чотири першорівневих стилю.

– структурний програмування (дії і умови локальні);

– програмування від станів (дії глобальні, умови локальні);

– програмування від подій (дії локальні, умови глобальні);

– сентенціальне програмування (дії і умови глобальні);

З програмістської точки зору ці стилі розрізняються за такими характеристиками.

У структурному програмуванні, якому зараз вчать як монопольному перворівневому стилю, керуючі структури утворюють ієрархію, а потоки передачі даних в принципі повинні погоджуватися з даної ієрархією. Математичною моделлю програм є тут обчислюваної функції. Логічною моделлю програмування є лінійна конструктивна логіка. Структури сучасних традиційних мов програмування (Pascal, C, Ada тощо.) підтримують насамперед даний стиль.

У програмуванні від станів процес представляється як зміна станів системи. Новий стан виникає в результаті дії, що змінює старе стан, а вибір цієї дії залежить від перевірки умов. Математичною моделлю програми служить кінцевий автомат. Логічною моделлю служить нільпотентной конструктивна логіка. Природним способом програмування такого завдання на сучасних мовах програмування є використання операторів goto або об'єктів, що обмінюються інформацією через загальне поле пам'яті.

У сентенціальному стилі (Рефал, Пролог) кожен крок програми перевіряє все поле зору на відповідність зразком, знаходить тим самим застосовне правило перетворення, і, згідно із знайденим правилом, перетворює все поле пам'яті.

У програмуванні від подій умова полягає в тому, що в системі відбулося деяке подія, кращим обробником якого виявилося дана дія. Математичні і логічні моделі такого підходу ще недостатньо развіти6. Як математичних моделей можна згадати мережі Петрі. Кандидатом на роль логічних моделей є релевантні логіки.

Високорівневе програмування в принципі повинно мати набір стилів, що відповідає тому ж морфологічному блоку. Це насамперед

функціональний стиль, найбільш яскравим і доведеним прикладом якого є LISP. Тут реалізовані обчислення над функціями та структури даних повністю відповідають структурі функцій. Математичним описом даного стилю є λ-числення. Логічною моделлю є інтуіційная логіка.

Об'єктно-орієнтований стиль є кроком до високорівневого стилю, відповідного інваріанта: «Дії глобальні, умови локальні». При діях цілком змінюються об'єкти, що представляють собою моделі, що складаються з даних і методів їх обробки. На жаль, немає теорії функціоналів вищих типів, що описують дії, які змінюють світ, а нільпотентна конструктивна логіка також не розроблена для формул вищих порядків.

Сучасна інформатика і на рівні теорії (теорія областей даних, яка виросла з завдання формалізації функціонального програмування і далеко переступила її межі), і на рівні практики підійшло до моменту, коли можливе створення сентенціального високорівневого стилю, стилю обчислень над обчисленнями. Високорівневий подієвий стиль поки що не підготовлений, але є деякі непрямі намітки того, що вдала формалізація програмування від подій одночасно дасть ідеї і для конструкцій вищих рівнів.

Жоден стиль програмування не описується суто логічно. Це пов'язано перш за все з наявністю в мовах програмування фіксованих типів даних, наприклад, чисел.

Потрібно чітко розділити композицію понять і їх конструювання. Роботу з вихідними типами даних потрібно поміщати в середовище суворих обмежень на правила композиції, а там, де потрібно створювати складні структури понять або дій, конкретні дані потрібно повністю відкинути, маючи лише непрямі посилання на них через використовувані модулі нижчого рівня. Отже, обчислення та міркування концептуально суперечать один одному.

З цього випливає і педагогічний висновок: математика, яка викладається на базі монополії аналізу, годиться для опису фізичних, але не годиться для опису інформаційних процесів. Для програмістів набагато

важливіше логіка, алгебра, топологія, лінгвістика і філософія. Для них важливіше перетворювати поняття, ніж формули.

Стиль структурного програмування найкраще обґрунтований теоретично (теорема Бема-Джакопіні про структуруванні схем програм), використовує теорію коректно, але навіть на ньому видно, наскільки погано «практики» сприймають навіть недвозначні і легко формулюються теоретичні висновки. Сприйнявши щось одне, вони вже не можуть сприйняти інше, пов'язане з ним, а кидаються відчайдушно реалізовувати перше.

По-перше, вже Бем і Джакопіні показали, що для структурування довільних схем часто потрібно додавати додаткові змінні, по суті справи, що запам'ятовують ознака того, що подією закінчилося, а потім багаторазово перевіряти їх. Д. Кнут показав, що для подолання цього недоліку досить ввести завершувач для блоків і мати право у внутрішньому блоці завершити будь-який загальний (так звані структурні переходи). Але ніде структурні переходи в коректному вигляді не реалізовані, і в результаті доводиться вводити оператори goto і довго і безсистемно пояснювати, коли ж їх пристойно використовувати. Вже Java могла б замість безсистемних синтаксичних обмежень на переходи просто скористатися чудово розробленої і абсолютно ясною концепцією завершувачів! За 30 років можна було б чогось і навчитися!

По-друге, проведені дослідження показали, що робити програму послідовною з самого початку означає спотворювати її логіку. Ніяких додаткових зусиль не вимагало б сприйняти хоча б концепцію спільно виконуваних операторів Алгола 68, коли детермінується лише те, що необхідно детермінувати! Але в першому наближенні зручно, щоб програма завжди виконувалася однозначно, це і хакерських трюкам сприяє.

І, нарешті, ніхто не здогадався, незважаючи на багаторазові спостереження (підтверджуються результатами теорії складності), що цикли і рекурсії погано змішувати разом. Насправді є два ізводу структурного програмування. Коли у нас дані також структуровані ієрархічно, то доцільно

користуватися циклами, а коли в типах даних є посилання на самих себе, доцільна рекурсія.

Є ще одна, менш очевидна, недоробка. Це практичне ігнорування того, що програму не опишеш поняттями, які задані всередині її самої. Давно вже відомо, що для повного опису властивостей програми необхідні примари - поняття, в програму не входять, часто просто шкідливі для обчислень і навіть не представимо в машині, але необхідні для коректного опису системи.

Програмування від станів або автоматне програмування застосовується тоді, коли завдання описується як сукупність станів світу і переходів між ними. Засоби для його підтримки вціліли у всіх традиційних мовах, і воно чудово реалізується як послідовність перевірок умов, викликів функцій, відповідних станам автомата, і переходів в черговий стан. Функції, що реалізують стану автомата, найчастіше природно програмуються структурному стилі (і тут ми бачимо перший приклад того, що стилі не повинні ходити поодинці).

Для сентенціального програмування є два альтернативних (і навіть теоретично несумісних) підходи - програмування, засноване на поверненнях і уніфікації (Prolog) і програмування, засноване на конкретизації (Рефал). Обидва вони живі. Перший варіант краще підходить для моделювання логіки рішень, коли ефективність не критична і явне виписування умов утомливо (завдання пошуку). Другий варіант краще підходить для синтаксичних перетворень ієрархічних структур.

Виключно цікавий приклад сентенціальной програмування в зв'язку з глобалізацією. Глобалізація посилює монополізацію, в тому числі і в інтелектуальній схемою. При множині спільних рис Рефал і Prolog несумісні за механізмом зіставлення поля зору з зразком і механізму управління. Якщо Рефал чудово пристосований для & -параллелізма, коли паралельно запускається кілька процесів і результат виходить узагальненням усіх отриманих приватних результатів, то Prolog настільки ж чудово пристосований до ∨-паралелізму, коли кілька процесів змагаються в тому,

хто швидше досягне мети. З огляду на «залізної завіси» обидві концепції вижили, встали на ноги, актуальні й досі.

Програмування на Prolog традиційно називають логічним. Від логіки тут нічого не залишилося, і термін вводить в оману (і навіть якщо б залишилося, логіка тут була лише інструментом, а ставити конкретний інструмент попереду концепції те ж саме, що віз попереду коня). Теорія була коректно застосована і конкретизована лише в разі РЕФАН (але з точки зору інтерфейсів цю мову так і залишився недопрацьованим, хоча з самого початку була ясна його спеціалізованість і, відповідно, доцільність використання в системах, де інші підзадачі вирішуються іншими методами).

Prolog набагато краще сполучається з іншими мовами, але безнадійно неефективний і по ресурсах, і технологічно для нетривіальних випадків. Додані примочки, які повинні були б підвищувати ефективність, повністю розвалили концепцію мови. Prolog яскраво демонструє, що відбувається, коли теорію використовують як заклинання, а не як обгрунтування. Хорновскіе диз'юнкт, що дали ідею мови, абсолютно неадекватні тому, що вийшло, коли взяту з логіки ідею уніфікації з'єднали з іншого, чудово поєднується з уніфікацією, ідеєю: непрямого управління за допомогою повернення після невдач. В результаті співтовариство Prolog наполегливо тримається за жупел логіки, що заважає йому усвідомити реальні досягнення підходу.

Програмування від подій (подієве програмування) виникає тоді, коли дії активізуються в зв'язку з тим, що виникла деяка стан системи і не знайшлося кращого кандидата для реакції на цю дію. Таким чином, тут обробник умов централізований, він визначає, хто готовий і хто повинен обробити подію, що відбулася, наприклад, клацання миші (щоб встановити контекст клацання, потрібно часом проаналізувати стан інтерфейсу всіх працюючих процесів).

Програмування від подій розглядає два підходи:

– власне подієвий програмування, коли подія (наприклад, клацання миші) перпедає процесу-обробнику важливу інформацію (наприклад, про

місце клацання);

– програмування від пріоритетів: подія полягає в тому, що всі менш пріоритетні процеси нічого не можуть зробити, і ніякої позитивної інформації керуючому процесу не дає.

Це програмування в самостійному вигляді реалізовано в деяких мовах скриптів для спеціалізованих процесорів. Але насправді це - рівноправний стиль, реалізований в традиційних мовах хакерськими прийомами.

Функціональний стиль програмування, коли програма являє собою функціонал високого рівня, що перетворює функції, представлений мовами LISP ML, Huskell і іншими. Якщо функції типізовані, то цей підхід, зберігаючи можливості понять вищих рівнів виключно компактно висловити складну структуру, до того ж ефективний по використанню ресурсів. Але нинішні реалізації функціонального програмування не можуть втриматися від використання рекурсій і нетіпізіруемих конструкцій типу оператора обчислення довільного виразу або оператора нерухомої точки, змішуючи тим самим один з ізводів структурного програмування з функціональним. Сміливості позбутися від нижнього рівня і від зайвих можливостей ніяк не вистачає. Це створило функціональному програмуванню репутацію вкрай неефективного стилю, відповідного лише для прототипів програм.

Більш того, в самому LISP чітко виділено і пристойними програмістами використовується концептуально цілісне ядро, а наступні мови з'явилися прикрашеним виродженням ідеї, оскільки вставляли в неї модні нові можливості без їх критичного аналізу. Дійшло до того, що в функціональне програмування для його 'посилення' вставляється концептуально незмірно більш низькі і зайві в розвиненому світі вищих сутностей елементи об'єктно-орієнтованого програмування.

Об'єктно-орієнтоване програмування є кроком до високорівневому програмування, відповідному функціональному для дій і подій. Об'єкти показали навіть досить неосвіченим людям, що для складних систем робота з діями ефективніше роботи з даними.

Задекларовано, що об'єкти виникли на базі теорії абстрактних типів даних, але тут ситуація ще важче, ніж в Prolog. Теорія просто не має ніякого відношення до практики в тих випадках, коли вживаються теоретичні терміни, вони вживаються помилково. Наприклад, збагачення алгебри називається її конкретизацією.

З точки зору практики, це кілька позначок різних стилів. Один з них бере початок скоріше в психології і штучному інтелекті, і успішно застосовується для масового виробництва програм, де зовнішня упаковка важливіше суті. Тут об'єкти в різних контекстах можуть вести себе зовсім по-різному, так що вони відповідають ролям людей.

Розгляд стилів призводить до висновку про згубний вплив на сучасну високорівневу практику наступних факторів.

По-перше, це ілюзія універсальності. Пора розглядати слова «універсальна система» як щось подібне філософського каменю. Ілюзія універсальності змушує вводити в систему можливості, згубним чином розширюють її. Перехід до наступного рівня понять неможливий без заборони багатьох методів, які діяли на попередньому рівні. Але програмуванні цьому перешкоджає ілюзія універсальності, конкретизує в забобон сумісності, який змушує при розвитку системи тягнути за собою шлейф концептуально несумісних застарілих понять. Випадки, коли від цього забобону відмовилися, лічені.

По-друге, це ігнорування негативних результатів, або, більш загальному вигляді, виключно позитивне мислення. Закривання очей на теоретичні попередження ніколи ні до чого доброго не приводить, а їх систематичне вигнання з навчальних курсів на догоду душевного комфорту і 'позитивності' шкідливо позначається на здібностях до творчого мислення, один з прийомів якого: перетворити мінус в плюс

По-третє, це неправильно зрозумілий прагматизм. З теорії вихоплюється що-небудь одне, і як можна швидше оформляється у вигляді практичної системи.

Ще однією стороною того ж прагматизму є рання стандартизація. Випадкові особливості недостиглої системи фіксуються в стандарті, і, оскільки кращі фахівці в цій галузі занадто часто мають хакерські нахили і звикли використовувати недоробки в якості нібито віртуозних прийомів, ці недоробки зводяться в ранг священної корови. Наприклад, такий статус набула помилка в реалізації алгоритму уніфікації в ранньому Prolog і конкретний спосіб реалізації повернень, який був там застосований.

1.2 Подієвий стиль побудови програмного коду

В даний час широке поширення отримав уніфікована мова моделювання (Unified Modeling Language, UML) – це стандартний інструмент для розробки «креслень» програмного забезпечення. Його можна використовувати для візуалізації, специфікації, конструювання та документування артефактів програмних систем [2].

В UML функціональні моделі дискретних систем (ДС) поділяються на чотири різновиди в залежності від набору об'єктів, що представляються графічно і є основними:

– моделі, орієнтовані на активності (схеми алгоритмів, мережеві моделі робіт), будемо називати їх моделями активностей;

– моделі, орієнтовані на стани (автоматні моделі), тобто моделі станів;

– моделі, орієнтовані на події (подієві мережеві моделі, подієві графи і алгоритми), так звані подієві моделі;

– змішані моделі, тобто моделі, що мають в якості основних більше одного динамічного об'єкта, орієнтовані на активності і стани (діаграми діяльностей) та орієнтовані на події і стани (мережі Петрі).

Кожен з видів моделей відображає певний світогляд, певний погляд на поведінку досліджуваної системи. При цьому при описі функціонування ДС використовується певний набір термінів (понять). Залежно від конкретної прикладної області зручним є той чи інший спосіб представлення.

Реальний світ складається з подій, події в нашому житті часом відбуваються одночасно і несподівано. Під подією мається на увазі якийсь значущий факт, локалізований у часі і просторі. В контексті кінцевих автоматів події використовуються для моделювання певного впливу, яке може викликати перехід з одного стану в інший.

До числа подій відносяться сигнали, виклики, витікання певного проміжку часу або зміна стану. Події можуть бути синхронними і асинхронними; їх моделювання - одна зі складових моделювання процесів і потоків.

Будь-яке явище, яке може мати місце в дійсності, може розглядатися як подія. Подія (event) – це опис суттєвого факту, що відбувся в певному часі і просторі. Отримання сигналу, збігання проміжку часу, зміна стану - це приклади асинхронних подій, які можуть відбутися в будь-який момент. Виклики – це, як правило, синхронні події, які використовуються для запуску якоїсь операції.

Визначимо базові поняття.

Подія – це опис суттєвого факту, локалізованого в часі і просторі. Стосовно до автоматів подія означає дію (сигнал), яка може викликати перехід з одного стану в інший.

Сигнал – це різновид події, при використанні якої повідомлення передається асинхронно від одного об’єкта до іншого.

Події можуть бути внутрішніми або зовнішніми. Зовнішні події передаються між системою і її дійовими особами (прикладами можуть служити натискання кнопки або переривання від датчика запобігання зіткнень), а внутрішні – між об'єктами, існуючими в самій системі (приклад – сигнал, що генерується при переповненні).

Можна моделювати чотири види подій: сигнали, виклики, витікання проміжку часу і зміни стану.

Повідомлення – це іменований об'єкт, який асинхронно надсилається одним об'єктом і приймається іншим. Сигнал (signal) являє собою

класифікатор для повідомлень і сам є типом повідомлення.

Сигнал може відправлятися в результаті виконання якоїсь дії в процесі переходу (зміни стану) в автоматі. Його можна змоделювати як повідомлення, передане між двома ролями при деякій їх взаємодії. При виконанні методу теж можуть передаватися сигнали.

Якщо подія сигналу являє собою його екземпляр, то під дією виклику (call event) розуміється отримання деяким об'єктом запиту на виконання операції над ним. Подія виклику може привести до переходу між станами в автоматі або викликом методу на цільовому об'єкті. У той час як сигнал є подією асинхронною, подія виклику зазвичай синхронна. Це означає, що, коли один об'єкт ініціює виконання операції на іншому об'єкті, у якого є свій автомат, управління передається від відправника одержувачу, спрацьовує відповідний перехід, потім операція завершується, одержувач переходить в новий стан і повертає управління відправнику. У тих випадках, коли відправник не потребує очікуванні відповіді, виклик може бути визначений як асинхронний.

Подія часу являє собою закінчення якогось проміжку часу. В UML така подія моделюється за допомогою ключового слова after (після), за яким слідує вираз, що визначає цей проміжок. Вираз може бути простим - наприклад, after 2 seconds (через 2 секунди), або складним - наприклад, after 1 ms since exiting Idle (через 1 мс після виходу зі стану Очікування). Якщо явно не вказано інше, відлік часу починається з моменту входу в поточний стан.

За допомогою події зміни описується зміна стану або виконання деякого умови. Подія зміни відбувається один раз при зміні значення умови з помилкового на справжнє (але не навпаки). Поки умова залишається істинною, подія не повторюється. У більшості систем, керованих подіями, події сигналів утворюють ієрархію.

Використовуючи взаємодію, можна моделювати поведінку спільноти спільно діючих об'єктів. За допомогою кінцевого автомата можна змоделювати поведінку окремого об'єкта. Автомат – це модель поведінки,

яка специфікує послідовність станів об'єкта, через які він проходить протягом свого життєвого циклу у відповідь на події, а також реакції на ці події.

Автомати використовуються для моделювання динамічних аспектів поведінки системи. Здебільшого цей процес має на увазі специфікацію життєвого циклу примірників класу, варіанти використання або системи в цілому. Ці екземпляри можуть реагувати на такі події, як сигнали, операції або витікання якогось періоду часу. Коли відбувається подія, в залежності від поточного стану об'єкта спостерігається певний ефект. Ефект – це специфікація поведінки, що реалізується всередині автомата. В кінцевому рахунку ефекти проявляють себе у виконанні дій, що змінюють стан об'єкта або повертають будь-яке значення. Стан об'єкта – це період часу, протягом якого він задовольняє заданим умовам, виконує якусь діяльність або очікує певного події.

Автомат – це опис поведінки, який специфікує послідовність станів, через які проходить об'єкт за час свого існування у відповідь на події, а також його реакцію на ці події. Стан – ситуація під час життя об'єкта, в якій він задовольняє заданим умовам, здійснює якусь діяльність або перебуває в очікуванні подій.

Подія – це специфікація значущого випадку, локалізованого у часі і просторі. Стосовно до автоматів подія – це сигнал, який може ініціювати перехід від одного стану до іншого.

Перехід – зв'язок між двома станами, яка вказує на те, що об'єкт в першому стані виконає певні дії і перейде в другий, коли трапиться відповідна подія і задана умова буде задоволена.

Діяльність (activity) – це неатомарний процес (який складається з більш простих компонентів) всередині автомата, який відбувається в даний момент.

Дія (action) – це обчислення, що виконується та в результаті якого змінюється стан моделі або повертається деяке значення.

Поведінка об'єкта, який повинен відповідати на асинхронні

повідомлення або його поточна поведінка залежить від минулого, найкраще описується за допомогою автоматів. Це стосується екземплярів класів, які можуть приймати сигнали, включаючи багато активні об'єкти. Фактично об'єкт, який приймає сигнал, але не має для нього переходу в свій поточний стан і не відкладає реакцію на сигнал в цьому стані, буде просто його ігнорувати. Іншими словами, відсутність переходу для сигналу не є помилкою; це означає, що в даній точці сигнал не представляє інтересу. Можливо також використовувати автомати для моделювання поведінки систем в цілому, особливо реактивних, які повинні відповідати на сигнали дійових осіб за межами системи.

Стан (state) – це ситуація в життєвому циклі об'єкта, в якому він задовольняє заданим умовам, здійснює якусь діяльність або очікує певного події. Об'єкт перебуває в тому чи іншому стані обмежений період часу.

Існує два особливих стану, які можуть бути визначені для автомата об'єкта. Перший з них – початковий стан, яке означає початкову точку, за замовчуванням встановлену для автомата. Другий стан - кінцевий; він означає, що робота автомата завершена.

Перехід – це зв'язок між двома станами, що означає, що об'єкт в першому стані повинен виконати певні дії і перейти в другий стан, коли відбудеться певна подія, і будуть задоволені задані умови. До тих пір поки перехід не відбувся, про об'єкт кажуть, що він знаходиться в початковому стані; після переходу він перебуває в цільовому стані.

Перехід складається з п'яти частин:

– початковий стан – стан, на який впливає перехід, якщо об'єкт знаходиться в початковому стані, то вихідний перехід може відбутися, коли об'єкт прийме подію, що ініціює перехід, і буде виконано захисна умова (якщо вона є);

– ініціююча подія – подія, яка, будучи розпізнаними об'єктом в початковому стані, ініціює виконання переходу, забезпечуючи істинність його захисного умови;

– захисна умова – логічне вираження, яке обчислюється при ініціюванні переходу після отримання події, якщо вираз істинний, то виконання переходу дозволяється, в іншому випадку перехід не виконується, і якщо немає інших переходів, які можуть бути ініційовані тим же подією, то подія втрачається;

– ефект – це поведінка, яка виконується (така як дія), яка може бути властива об'єкту, який володіє даними автоматом, і опосередковано – з іншими об'єктами, які видимі даному;

– цільове стан – стан, в якому виявляється об'єкт після завершення переходу.

Внутрішній перехід (internal transition) – це такий перехід, який реагує на подію відповідним ефектом, не змінюючи стану. Внутрішній перехід символізується включенням рядка переходу (разом з ім'ям події, необов'язковим захисним умовою і ефектом) в символ стану замість стрілки переходу. Всякий раз, коли об'єкт знаходиться в деякому стані і відбувається подія, яким позначений внутрішній перехід, виконується відповідний ефект, але об'єкт при цьому не залишає стану і не повертається в нього повторно. Таким чином, подія обробляється без виклику дій виходу і входу.

Коли об'єкт знаходиться в якомусь стані, він зазвичай простоює, чекаючи настання події. Однак іноді стає необхідно змоделювати якусь діяльність, виконувану в цьому стані (do-activity). Іншими словами, перебуваючи в стані, об'єкт може що-небудь робити до тих пір, поки це не буде перервано подією. Якщо поява події викличе перехід, який призведе до виходу зі стану, будь-яка виконувана в стані діяльність негайно припиниться.

Розглянуті властивості станів і переходів вирішують цілий ряд типових проблем моделювання автоматів. Однак у автоматів є властивість, яке дозволяє ще більше спростити моделювання складного поведінки, - підстан (substate), тобто такий стан, яке входить до складу іншого.

Простий стан – це такий, який не має внутрішньої структури. Стан, що має вкладення, тобто підстани, називається складним (композитним).

Підстани можуть бути паралельними (ортогональними) або послідовними (неортогональноми), глибина вкладення станів не обмежена.

Найбільш загальне призначення автоматів – моделювання життєвого циклу об'єкта, особливо якщо мова йде про примірники класів, варіантів використання і про систему в цілому. У той час як взаємодія моделює поведінку спільноти працюють разом об'єктів, автомат моделює поведінку одного об'єкта протягом його життєвого циклу - наприклад, так, як це буває з користувача інтерфейсами, контролерами і різними технічними пристроями. Коли моделюється життєвий цикл об'єкта, доводиться явно специфікувати три моменти: події, на які об'єкт може реагувати, безпосередню реакцію на ці події і вплив більш раннього поведінки на поточний. Крім того, моделювання життєвого циклу об'єкта включає в себе прийняття рішень про порядок, в якому об'єкт може осмислено реагувати на події, починаючи з моменту його створення і закінчуючи знищенням.

Щоб змоделювати життєвий цикл об'єкта, необхідно:

– визначити контекст автомата - чи буде це клас (пристрій) або система в цілому;

– встановити початковий і кінцевий стани об'єкта, щоб управляти решті моделлю, можливо, знадобиться встановити перед- і постумови початкового і кінцевого станів відповідно;

– прийняти рішення щодо подій, на які повинен реагувати об'єкт. Їх можна виявити в інтерфейсах об'єкта (якщо вони вже специфіковані); якщо ж ні, слід розглянути, які об'єкти можуть взаємодіяти з даними в наявному контексті і які події вони можуть передавати і приймати;

– від початкового стану до кінцевого виділити ті стану вищого рівня, в яких може перебувати об'єкт, з'єднати їх переходами, ініційованими відповідними подіями, продовжувати роботу, додаючи дії до цих переходах;

– ідентифікувати вхідні і вихідні дії (особливо якщо виявиться, що відповідна ідіома використовується в автоматі);

– при необхідності розширити виділені стану підстанів;

– перевірити, чи всі згадані події в автоматі відповідають подіям, очікуваним в інтерфейсі об'єкта, аналогічним чином перевірити, чи всі події, очікувані інтерфейсом об'єкта, обробляються в автоматі;

– переконатися, що всі дії, згадані в автоматі, підтримуються зв'язками, методами і операціями об'єкта;

– провести трасування автомата (або вручну, або з застосуванням інструментальних засобів), щоб перевірити його на відповідність очікуваним послідовностям подій і реакцій на них. Особливо старанно слід пошукати недосяжні стану і стану, в яких автомат може зациклитися;

– після реорганізації автомата знову перевірити його на предмет того, чи не змінилася семантика об'єкта.

При моделюванні автоматів на UML слід пам'ятати, що кожен з них виражає динамічні аспекти окремого об'єкта, зазвичай представляє екземпляр класу, варіанти використання або систему в цілому.

Добре структурований автомат повинен мати наступні характеристики:

– бути досить простим і не включати зайвих станів або переходів;

– завдяки ясному контексту мати доступ до всіх об'єктів, від основного об'єкту (всі ці сусіди повинні використовуватися тільки при необхідності забезпечення поведінки, специфікованого автоматом);

– бути ефективним, тобто реалізовувати свою поведінку;

– бути зрозумілим, а тому називати свої стани і переходи в термінології словника системи;

– не допускати занадто глибоких рівнів вкладеності (вкладених підстанів першого-другого рівнів цілком вистачить для опису самого складного поведінки);

– помірно використовувати ортогональні області, оскільки часто переважно застосування активних класів.

При зображенні графої моделі автомата необхідно:

– уникати переходів, що перетинаються;

– розкривати складові стану «за місцем» тільки в тому випадку, якщо це зробить діаграму зрозумілішою.

У системах логічного управління, процес керування здійснюється не тільки сигналами від об’єкта управління а й зовнішніми подіями. При цьому використовуються принципи обробки подій, які достатньо докладно розроблені в різних алгоритмічних мовах програмування.

Подія - це абстрактне поняття, яке характеризує таку зміну зовнішнього середовища, яка обумовлює певну реакцію системи.

У програмних системах у загальному випадку подіями можуть бути:

– графічна взаємодія користувача з комп'ютером (з використанням допомогою миші, клавіатури, сенсорного екрана);

– виконання операцій введення-виводу;

– взаємодія користувача з сервісами Інтернет;

– розпізнавання образів в зображеннях;

– реакції систем Internet of Things на поточну обстановку;

Прийнято події реального світу умовно ділити на виключення (аварійні події) і прості події - дії користувача (на клавіатурі, миші), повідомлення інших програм і потоків.

Обробка подій подібна до обробки виключень. У обох випадках можуть бути задіяні обробники. Хоча виключення можуть бути генеровані або явно програмним кодом, або неявно апаратними засобами, події створюються зовнішніми до системи управління діями. Вважається, що більшість подій викликана взаємодією користувача з елементами графічного інтерфейсу користувача, наприклад кнопками. Реакція на такі взаємодії з користувачем є найбільш поширеною формою обробки зовнішніх подій.

Широке застосування подієво-орієнтованого програмування обумовлене найчастішим використанням графічних інтерфейсів. Багато веб-документів мають динамічні характеристики. Такий документ може надавати, наприклад, форму замовлення для користувача, який обирає товар, клацаючи по кнопках клавіатури або натискаючи на зображення на

сенсорному екрані. Необхідні внутрішні обчислення, пов'язані з цими діями користувача, виконуються обробниками подій, які реагують на події клацання.

У програмуванні на основі подій виділяють п'ять елементів:

- джерело події;
- сигнал події;
- слухач події і диспетчер;
- обробник події;
- об'єкт події.

Конкретна подія має бути зафіксована, його значення має бути визначене. і якщо воно є бажаною подією, може бути зроблена дія. Джерело події генерує сигнал події. Наприклад, при клацанні мишею по гіперпосиланню або прапорцю, миша є джерелом подій, точніше, відповідний їй програмний компонент. Джерело події повинно мати методи, що дозволяють реєструвати слухачів своїх подій і виключати їх з цього списку. Таким чином, джерело події зберігає список зареєстрованих слухачів подій, щоб мати можливість повідомляти тільки ним про подію, що сталася.

Слухач події приймає сигнал і передає його характеристики (як об'єкт події) через адаптер в потрібному обробнику події, що виконує відповідні дії (залежно від стану системи). Якщо слухач обслуговує декілька адаптерів, він додатково вирішує задачу диспетчера (комутує порядок обробки сигналу). Обробка події ускладнюється, коли декілька подій спільно використовують загальний програмний або апаратний ресурс.

Обробники подій реалізуються з використанням анонімних об'єктів і приватних внутрішніх класів, до яких користувач не має прямого доступу.

У контур обробки події може включатися адаптер. Адаптер події це інтерфейсі з порожніми методами, які відповідають подіям. Порожні методи перевизначаться конкретними класами обробників.

Обробка подій, як правило, пов'язана з системами реального часу, які визначаються, виходячи з динамічних характеристик процесів об’єктах

керування і пов'язаних з ним подій. При цьому під подією розуміється така зміна зовнішнього середовища, яке обумовлює конгкретну реакцію системи. Наприклад, зміна атмосферного тиску є подією для барометра, але не є подією для секундоміра. Подія - абстрактне поняття: події можуть генеруватися як зовнішнім середовищем. так і всередині самої системи її компонентами. Приклад зовнішнього події - натискання на кнопку. Затримка, з якою система повинна обробити це натискання, і стане вихідною вимогою реального часу. У свою чергу, і механічний вплив на кнопку призводить до виникнення вторинних внутрішніх подій – зміни напруги на одному з портів мікроконтролера.

Одні події можуть породжувати інші події більш високого рівня абстракції. Наприклад, в результаті обробки низкорівневої події – зміни напруги в колі кнопки, блоком фільтрації брязкоту генеруються події середнього рівня – натиснення і відпуск кнопки. У свою чергу, на їх основі можна згенерувати високорівневі події - подвійне натискання або натискання з повтором, якщо кнопка утримується тривалий час.

У мікроконтролерних системах управління обробка подій зазвичай здійснюється програмно. Для такої обробки характерні два моменти: по-перше, при обробці події система протягом певного часу виявляється несприйнятливою до нових аналогічних подій, а по-друге, сама обробка являє собою послідовність інструкцій процесора, кожна з яких виконується послідовно, в темпі, що задається тактовою частотою. Таким чином, програмно-керована система функціонує завжди дискретно. Причому мінімально можливий інтервал дискретності - одна інструкція процесора. Такі системи відносяться до класу кінцевих динамічних систем.

Окремим випадком систем з обробкою переривань є так звані foreground/background-системи, які включають основну програму типу «петля» (background) або безкінечний цикл і обробники переривань (foreground). Оскільки обробка переривань реалізована апаратно, такі системи вимагають мінімальних витрат для організації багатозадачності.

Взаємне виключення подій зазвичай реалізується шляхом часового блокування переривань, а для збереження контексту при виклику обробників використовується єдиний стек основної програми з урахуванням наявної вкладеності переривань.

1.3 Автоматне програмування

В останні роки велика увага приділяється розробці технологій програмування для вбудованих систем і систем реального часу, до яких пред'являються високі вимоги щодо якості програмного забезпечення.

Якщо в програмуванні останнім часом все ширше використовується поняття «подія», то запропонований підхід базується на понятті «стан». Тому область програмування, що базується на понятті «стан», була названа «автоматне програмування», а процес створення таких програм - «автоматне проектування програм» [3].

Особливість даного підходу полягає в тому, що при його використанні автомати задаються графами переходів, для розрізнення вершин в яких вводиться поняття «кодування станів». При виборі «багатозначного кодування» за допомогою однієї змінної можна розрізнити стани, число яких збігається зі значністю обраної змінної. Це дозволило ввести в програмування таке поняття, як «спостережність програм».

В рамках запропонованого підходу програмування виконується «через стани», а не «через змінні», що дозволяє краще зрозуміти і уточняти завдання і її складові частини. При цьому необхідно відзначити, що в автоматному програмуванні налагодження проводиться шляхом протоколювання в термінах автоматів.

В силу того, що в рамках цього підходу від графа переходів до тексту програми пропонується переходити формально і ізоморфно, то при застосуванні мов програмування високого рівня це найбільш раціонально виконувати за допомогою конструкції `switch` мови Сі або її аналогів в

інших мовах програмування (`case` для VHDL).

Надалі автоматний підхід був поширений на подієві системи, які називаються також «реактивними». У них зазначені вище обмеження зняті. Як випливає з назви цих систем, в них серед вхідних впливів використовуються події, як вихідних впливів застосовуються довільні процедури, а в якості операційних систем - будь-які операційні системи реального часу. Для програмування подієвих систем із застосуванням автоматів був використаний процедурний підхід, і тому таке програмування було названо «програмування з явним виділенням станів»

При цьому вихідні сигнали «прив'язані» до дуг, петель або вершин графів переходів (застосовуються змішані автомати – автомати Мура-Мілі). Це дозволяє в компактному вигляді представляти послідовності дій, які є реакціями на відповідні вхідні сигнали.

Особливість запропонованого підходу до програмування цього класу систем полягає в тому, що в них підвищується централізація логіки за рахунок усунення її в обробниках подій і формування системи взаємопов'язаних автоматів, які викликаються з обробників.

Інша найважливіша особливість описуваного підходу полягає в тому, що при його застосуванні автомати використовуються триєдиний: при специфікації; при програмуванні (зберігаються в програмному коді); при протоколюванні, що виконується, як зазначено вище, в термінах автоматів. Останнє дозволяє контролювати правильність функціонування системи автоматів. Протоколювання виконується автоматично по побудованій програмі і може використовуватися для задач великої розмірності при складній логіці програми. Автоматний підхід пропонується застосовувати не тільки при створенні системи управління, але і при моделюванні об'єктів управління.

Для систем логічного управління при введенні вхідних параметрів по запиту, запропонована SWITCH-технологія, призначена для алгоритмізації і програмування завдань логічного управління [4].

Синхронність програмної реалізації алгоритмів логічного управління робить автоматні моделі на відміну від асинхронної апаратної реалізації, яка застосовується до теперішнього часу в системах управління, адекватно здійсненними, що забезпечує можливість широкого застосування таких моделей на практиці в цьому випадку.

Основним для автоматів є поняття «стан», що є абстрактним поняттям, що відбиває деяку стадію реалізації алгоритму. Для надання стану фізичного сенсу в більшості випадків пропонується пов'язувати з ним одну з комбінацій значень вихідних змінних і технічний стан об'єкта управління. При цьому число станів в автоматі має бути не менше числа різних комбінацій вихідних змінних, які він повинен формувати. У найпростіших автоматах кожній комбінації вихідних значень відповідає один стан, а для більш складних автоматів одна і та ж комбінація може формуватися в різних станах. Ці стани можуть розрізнятися не введенням додаткових розрядів (довічних внутрішніх змінних) в вектор вихідних змінних, а присвоєнням станів десяткових номерів - багатозначних кодів (однією багатозначною внутрішньої змінної).

При цьому відсутня необхідність встановлювати і скидати велике число двійкових внутрішніх змінних, так як для кожного графа переходів вони можуть бути замінені лише однією багатозначною змінною, попереднє значення якої скидається автоматично при присвоєнні їй нового значення. Ця змінна приймає число значень, яка дорівнює кількості станів автомата, що різко підвищує наочність опису поведінки автоматів і суттєво відрізняє цей підхід від використовуваних в логічному проектуванні схем.

Функціонування автомата відбувається наступним чином: визначається стан, в якому він знаходиться; визначаються значення вхідних змінних; обчислюється наступний стан автомата. Місце формування комбінації значень вихідних змінних в цій послідовності дій визначається типом автомата, а наявність або відсутність затримки на програмний цикл їх зміни щодо зміни значення змінної стану – родом автомата.

При цьому в якості основної структурної моделі автомата в системах логічного управління пропонується використовувати автомат Мура другого роду, в якому значення вихідних змінних залежать лише від стану автомата, але не залежать від вхідних сигналів, а визначена затримка відсутня.

Важлива особливість пропонованого підходу полягає в тому, що в більшості випадків в кожному стані вказуються значення всіх вихідних змінних, а застосування замовчувань забороняється. При цьому різко спрощується внесення змін, так як значення вихідних змінних не залежать від передісторії, тобто значень вихідних змінних в інших станах.

Поведінку (динаміку) автоматів доцільно визначати (описувати) за допомогою графів переходів, що складаються з двох позначених компонент вершин і дуг. Вершини відповідають станам автомата Мура і позначаються дробом, чисельник в якій є номер стану, а знаменником - комбінація значень всіх вихідних змінних, що формуються в цьому стані.

Кожна дуга в графі переходів визначається булевою формулою, при рівності одиниці якої здійснюється перехід з вершини, відповідної початку дуги, в вершину, відповідну кінця дуги. При цьому початок і кінець дуги можуть збігатися і тоді їм буде відповідати одна вершина. У цьому випадку говорять, що має місце петля, що забезпечує збереження стану автомата доти, поки виконується умова, її позначати, а вершину з петлею називають стійкою. Кожна вершина без петлі є нестійкою - в ній автомат знаходиться тільки один програмний цикл. У загальному випадку і в стійкій вершині автомат також може знаходитися тільки один програмний цикл.

Символічне позначення пам'яті у вигляді вершини графа переходів більш наочно в порівнянні з тригером у функціональній схемі, для розуміння роботи якого необхідно знати правила його функціонування. Булева формула, яка відповідає кожній дузі, залежить не від усіх вхідних змінних автомата, а тільки від тих з них, які забезпечують розглянутий перехід. Тому якщо в деякій вершині змінити значення змінної, не зазначеної ні на одній з вихідних з цієї вершини дуг, то автомат свій стан не змінить.

Граф переходів чітко описує поведінку автомата (динаміку його функціонування) в термінах станів. Граф переходів, побудований певним чином, досить просто читається, по-перше, тому що його дуги, що виходять з кожної вершини, помічені не всіма змінними, а лише тими з них, які забезпечують переходи в сусідні вершини, по-друге, по графу переходів явно видно, які значення і в якому порядку повинні приймати будь змінні для забезпечення переходів з даної вершини в будь-яку із заданих, і по-третє, тому що в ньому відсутні прапори, а в його вершинах вказані значення всіх вихідних змінних, виключаючи тим самим їх залежність від передісторії. Це дозволяє читати граф переходів на відміну, наприклад, від функціональної схеми, по якій доводиться вважати, так як вона не містить в явному вигляді значень вихідних змінних.

Граф переходів не має містити генеруючих контурів, а для кожної його вершини повинні бути забезпечені повнота і несуперечність. Повнота розуміється в тому сенсі, що для будь-якого вхідного набору змінних, що позначають вихідні з цієї вершини дуги, визначено перехід в одну з вершин графа. У математичній формі умова повноти виражається так: диз'юнкція позначок всіх вихідних з вершини дуг повинна бути тотожно дорівнює одиниці. Несуперечливість розуміється в тому сенсі, що не повинно існувати жодного вхідного набору, при якому виконуються умови, що забезпечують можливість переходу більш ніж в одну вершину графа переходів. У математичній формі це виражається наступним чином: кон'юнкція позначок двох будь-яких вихідних з вершини дуг повинна бути тотожно дорівнює нулю.

Забезпечення несуперечності є необхідним, а повнота в явному вигляді може вимагатися не для всіх алгоритмічних і програмних моделей. Досягнення повноти в зазначеному вигляді в ряді випадків є досить трудомістким, але буває необхідним, наприклад для отримання системи булевих функцій, еквівалентній графу переходів.

При реалізації графа переходів за допомогою конструкції switch її структура дозволяє забезпечити перше із зазначених властивостей тільки розстановкою пріоритетів для всіх помічених дуг, що виходять з кожної вершини графа, а друге – без позначки петель. Це істотно спрощує перехід від специфікації до програмної реалізації алгоритмів при застосуванні мов високого рівня, що містять в своєму складі зазначену конструкцію або її аналог. При використанні графа переходів по номеру його вершини (багатозначного коду стану) можна визначити, де знаходиться автомат в даний момент часу в просторі станів, які значення при цьому приймають всі його вихідні змінні і які вхідні змінні і при яких значеннях забезпечують перехід в наступний стан. Без застосування поняття «стан» визначити «місцезнаходження» в зазначеному просторі автомата з пам'яттю, що формує в тому числі і однакові комбінації значень вихідних змінних, неможливо. З викладеного випливає, що графи переходів доцільно застосовувати в якості мови специфікації. Застосування цієї мови спільно з графами досяжних маркувань дозволяє проводити детальний аналіз алгоритмів до їх програмування. При використанні інших мов, відмінних від графів переходів, для специфікації зазначених алгоритмів виникають проблеми з тестуванням і внесенням змін, описані в наведеній вище історії.

Графи переходів в якості мови специфікацій можуть застосовуватися не тільки для опису автоматів, які реалізують алгоритми логічного управління, але і при створенні моделей об'єктів, керованих цими автоматами, що дозволяє з єдиних позицій описувати комплекс «КА-ОА».

В якості мови реалізації графів переходів може бути обрана практично будь-яка мова програмування. Серед алгоритмічних мов високого рівня найбільш доцільно вибрати той з них, який містить конструкцію, що дозволяє ізоморфно відображати графи переходів в текстах програм. У разі існування такої конструкції з'являється можливість проводити перевірку коректності переходу від специфікації до програми не в динаміці (тестуванням), а в статиці - звіркою тексту програми з графом переходів.

Як зазначалося вище, такою конструкцією є, наприклад, конструкція switch, що входить до складу мови С. Існують різні варіанти реалізації графа переходів за допомогою конструкції switch. Розглянемо один з них стосовно до автоматів Мура першого роду. При цьому графу переходів присвоюється змінна Y, число значень якої дорівнює числу вершин s в графі. Число міток case, що входять до складу конструкції switch, в цьому випадку дорівнює s. У мітці case, відповідної і-й вершині графа, в першому рядку перераховуються вихідні змінні і прийняті ними значення, а в останньому рядку вказується оператор break. Між першою і останньою рядками розміщуються оператори де – булева формула, позначати дугу між вершинами з номерами Число операторів if в мітці case дорівнює числу що виходять з даної вершини дуг (без урахування петлі). При наявності в вершині петлі вона реалізується зазначеним вище оператором break, який забезпечує також і вихід з оператора switch після реалізації не більше ніж одного переходу в графі переходів. При такій організації програми забезпечення несуперечності і повноти може бути виконано тільки розстановкою пріоритетів. При цьому формула з найбільшим пріоритетом повинна розміщуватися в останньому операторі if.

Запропонована стандартна реалізація алгоритмів логічного управління, що базується на використанні конструкції switch, забезпечує лінійну залежність складності програми від параметрів вихідного опису (специфікації), що надзвичайно важливо для забезпечення можливості широкого практичного застосування пропонованого підходу.

Таким чином, можна зробити висновок, що граф переходів автомата (state diagram) є не тільки візуальним відображенням алгоритму функціонування, але й повною його математичною моделлю.

2 МЕТОДИ ПРОЕКТУВАННЯ АВТОМАТНИХ СИСТЕМ ЛОГІЧНОГО УПРАВЛІННЯ

2.1 Особливості побудови автоматних систем 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 [5]. Загальна структура системи автоматичного управління показана на рис. 2.1.

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.

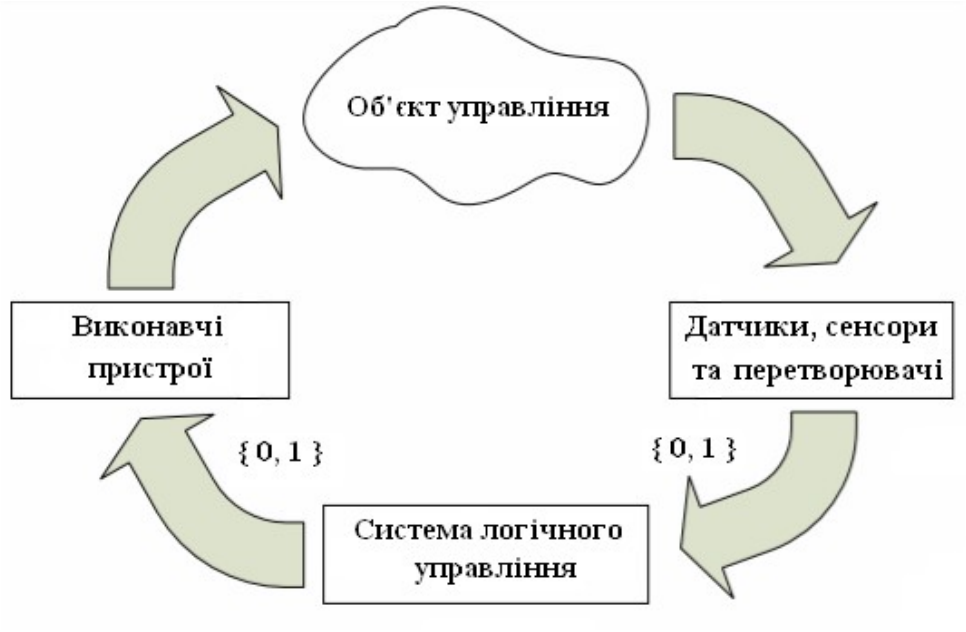


Рисунок 2.1 – Загальна структура системи автоматичного управління

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. 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.

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.

2.2 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 сповіщувальних 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 автоматному часі, який визначається тактами роботи автомата. Але більшість реальних систем логічного управління взаємодіють із зовнішнім світом в метричному часі, 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-який локальний цифровий пристрій, що реалізує алгоритм обробки інформації або управління, може бути реалізований 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 програмовані логічні інтегральні схеми (ПЛІС). Перевагою такого підходу є апаратна гнучкість (можливість реалізувати будь-який алгоритм) і досить велика швидкодія.

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) [6]. Часова змінна (timed variable) постійно збільшує своє значення і «скидається» в 0 при приході вхідного сигналу і переході автомата в новий стан. Часові змінні вимірюються в автоматних тактах.

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 .

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 багато модифікацій таких моделей, які по різному враховують як спосіб обробки подій, так і спосіб врахування затримок в станах автомата. Виходячи з особливостей функціонування систем логічного управління запропонована повна модель структурного часового автомату може бути представлена дев’яткою $W = \left(X, Y, Z, f, g, z_0, T_c, T_{to}, T_d \right)$, 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 – макси-мальна 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 автоматному часі, який вимірюється в автоматних тактах, тобто дискретних відрізках часу, за який автомат переходить з одного стану до іншого. 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 [7]. На рис 2.2 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.

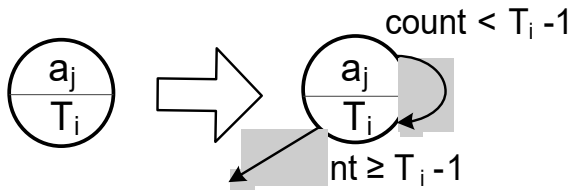


Рисунок 2.2 – Реалізація затримок у темпоральному графі переходів автомата Мура

На рис. 2.3 наведено фрагмент темпорального графа переходів автомата Мура для трьох станів. На рис. 2.4 наведено фрагмент часової діаграми функціонування даного автомата.

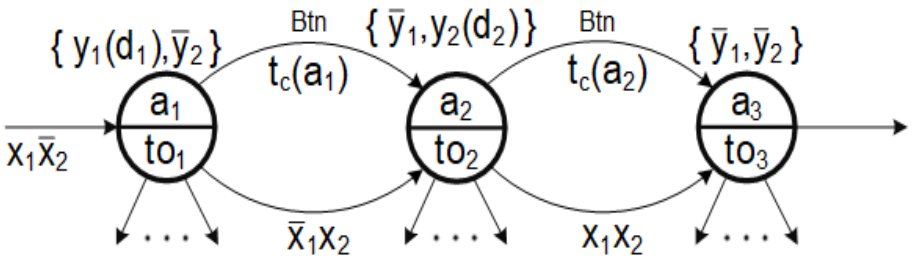


Рисунок 2.3 – Фрагмент темпорального графа переходів автомата Мура

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

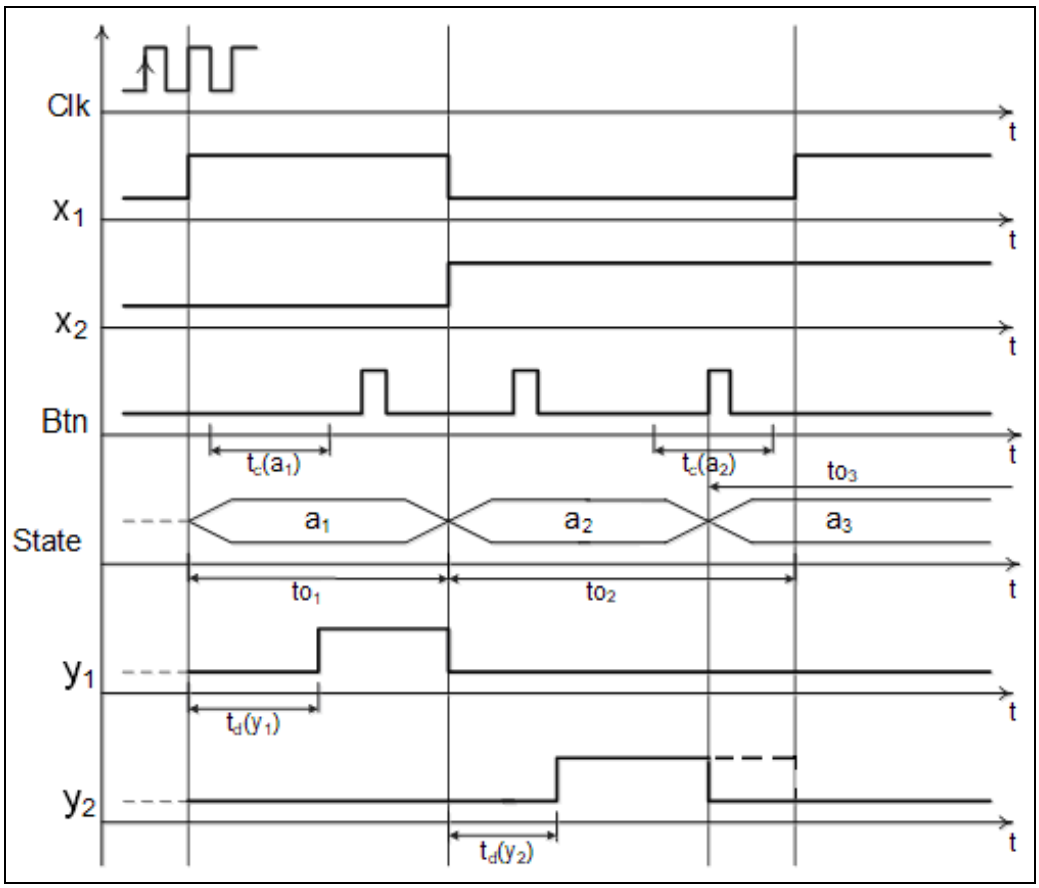


Рисунок 2.4 – Часова діаграма функціонування часового автомата

У загальному випадку, часовий автомат може містити усі три часові параметри < t_c, t_{to}, t_d >, але для конкретної задачі можуть використовуватися часові автомати з одним або двома із зазначених параметрів.

2.3 Класифікація HDL-моделей автоматних пристроїв керування

Для апаратної реалізації пристроїв логічного управління використовується модель структурного автомата. У структурних моделях вхідний алфавіт X абстрактного автомата перетворюється у множину вхідних сигналів (set input value), алфавіт станів A перетворюється у множину внутрішніх змінних Z, а вихідний алфавіт Y – у множину вихідних сигналів (set input value). При цьому всі три множини є кінцевими. У системах реального часу структурний автомат представляється моделлю часового автомата (timed FSM).

Стан структурного керуючого автомата в системах логічного управління характеризується сукупністю вихідних керуючих сигналів, які з'являються в певні моменти часу і мають певну тривалість. Переходи зі стану в стан визначаються типом вхідних сигналів і часом їх появи. За способом формування вихідних сигналів керуючі автомати класифікуються на моделі Мура і Милі, а за способом обробки вхідних сигналів на активні і пасивні.

У часового автомата Мура перехід зі стану в стан здійснюється «миттєво», а вихідні сигнали у поточному стані формуються з урахуванням затримок їх появи. При нульовій затримці вихідні сигнали з'являються в момент входу автомата в стан і не змінюються до кінця перебування автомата в даному стані.

У часового автомата Мілі час появи вихідних сигналів визначаються часом появи вхідних сигналів (з урахуванням затримок). Час існування вихідних сигналів автомата Мілі визначається тривалістю переходу в новий стан і в момент входу в новий стан вихідні сигнали даного стану обнуляються.

Якщо у автомата одночасно присутні вихідні сигнали, характерні для моделей Мура і Милі, то такий автомат має назву суміщений (С-автомат) [8].

Активний часовий автомат функціонує в залежності від значення вхідного сигналу в певний момент часу (не від зміни вхідного сигналу, тобто вхідного події). Зміна вхідного сигналу (вхідні дії) безпосередньо не ініціює зміни стану автомата. Автомат опитує вхідні сигнали в моменти часу, які визначаються алгоритмом його роботи і, таким чином, реалізує функцію переходів. Час переходу в новий стан у активного автомата фіксований (визначається різними затримками на різних переходах), тобто статичний.

Робота пасивного автомата визначається вхідною подією (автомат реагує не на значення вхідного сигналу в певний момент часу, а на подію в певний проміжок часу). Зміна вхідного сигналу (подія) безпосередньо ініціює перехід автомата в новий стан (реалізує функцію переходів) і визначає

момент появи вихідного сигналу (реалізує функцію виходів). З цієї точки зору пасивний автомат можна визначити як виключно подієвий. Час переходу в новий стан у пасивного автомата не фіксований, тобто динамічний. Він залежить від часу появи вхідної події, яка по суті є випадковою подією, в часовому проміжку очікування даної події.

Мікропрограмний автомат – завжди активний автомат: з кожним новим тактом Clk він переходить (по фронту Clk) в новий стан. В який саме стан перейде автомат, визначається вхідним сигналом, значення якого має бути стабільним протягом певного часу перед наступним фронтом Clk (для виконання коректного переходу, визначеного алгоритмом роботи автомата).

Вхідні події можна класифікувати як такі, що ініціюють або переривають (аварійні). Події, що ініціюють, запускають перехід автомата в новий стан і видачу відповідних вихідних сигналів. Подія, що перериває (аварійна), перериває перехід у автомата Мілі або перериває затримку в стані автомата Мура. Інакше кажучи, подія що перериває, змінює значення вихідного сигналу до його «стандартного» закінчення. Але при цьому деякі події, що ініціюють, можна розглядати як вхідні дії, наприклад включення живлення або запуск будь-якого процесу в керованій системі.

Якщо у структурного автомата частина вхідних сигналів розглядається як вхідні дії (які опитуються), а частина вхідних сигналів розглядається як події, то такий автомат прийнято називати змішаним. При виборі типу керуючого автомата в системах логічного управління слід враховувати обидва зазначених параметра.

Будь-який активний автомат, де є подія, що перериває, по суті, є змішаним, а крім того, певний вхідний сигнал може одночасно розглядатися як вхідна дія або подія, що разом з тим не порушує наведеної класифікації.

Якщо розглядати автомати по класифікації по вихідним сигналам, то автомати Мура, як правило, розглядаються як активні або змішані. Такі моделі складають більшу частину керуючих автоматів в системах логічного управління, тому що досить просто встановити відповідність між технічними

станами керованого об'єкта і набором керуючих сигналів в станах керуючого автомата Мура. Пасивні автомати Мура розглядаються досить рідко.

Розглянемо різні варіанти моделей часових автоматів Мілі, які реалізуються як активні, так і пасивні в залежності від класу вирішуваних завдань. Порівняння атрибутів активних і пасивних автоматів моделі Мілі наведені в табл. 2.1

Таблиця 2.1. – Класифікація атрибутів моделей автоматів Мілі

Атрибути моделі	Активна модель автомата Мілі	Пасивна (подієва) модель автомата Мілі
Вхідний сигнал X	Логічний сигнал X = {0, 1}, що змінюється в довільним-ний момент часу	Випадкова подія (зміна вхідного сигналу) в будь-який момент в інтервалі таймауту to
Функція переходів f	Вхідний сигнал фіксується в момент першого фронту синхросигналу в стані і ініціюється початок переходу довжиною to. Час перебу-вання в стані визначається таймаутом to	Поява події в інтервалі таймауту ініціює початок переходу, довжина якого не фіксована і визначаться довжиною вихідного сигналу Δt
Зовнішня (перериває-юча) подія Btn	Одиночний імпульс довжиною не менше періоду Clk в інтервалі t_c (в періоді to), який перериває перехід	Одиночний імпульс довжиною не менше періоду Clk в інтервалі t_c (в періоді Δt), який перериває перехід, і як наслідок , вихідний сигнал
Вихідний сигнал Y	Логічний сигнал Y = {0, 1}, що з'являється з затримкою t_d від початку переходу і закінчується із закінченням таймаута t_{to}	Логічний сигнал Y = {0,1}, в момент фіксації вхідної ініціюючого події X і має фіксовану довжину Δt
Функція	Вихідна затримка t_d	Довжина вихідного сигналу Δt

виходів g	знаходиться в інтервалі to і вихідний сигнал та таймер обнуляються з закінченням переходу	визначає загальну тривалість переходу (час знаходження в стані) і вихідний сигнал обнуляється із закінченням переходу

На рис.2.5 наведена загальна класифікація керуючих автоматів логічного управління в залежності від способів обробки вхідних сигналів та формування вихідних сигналів.

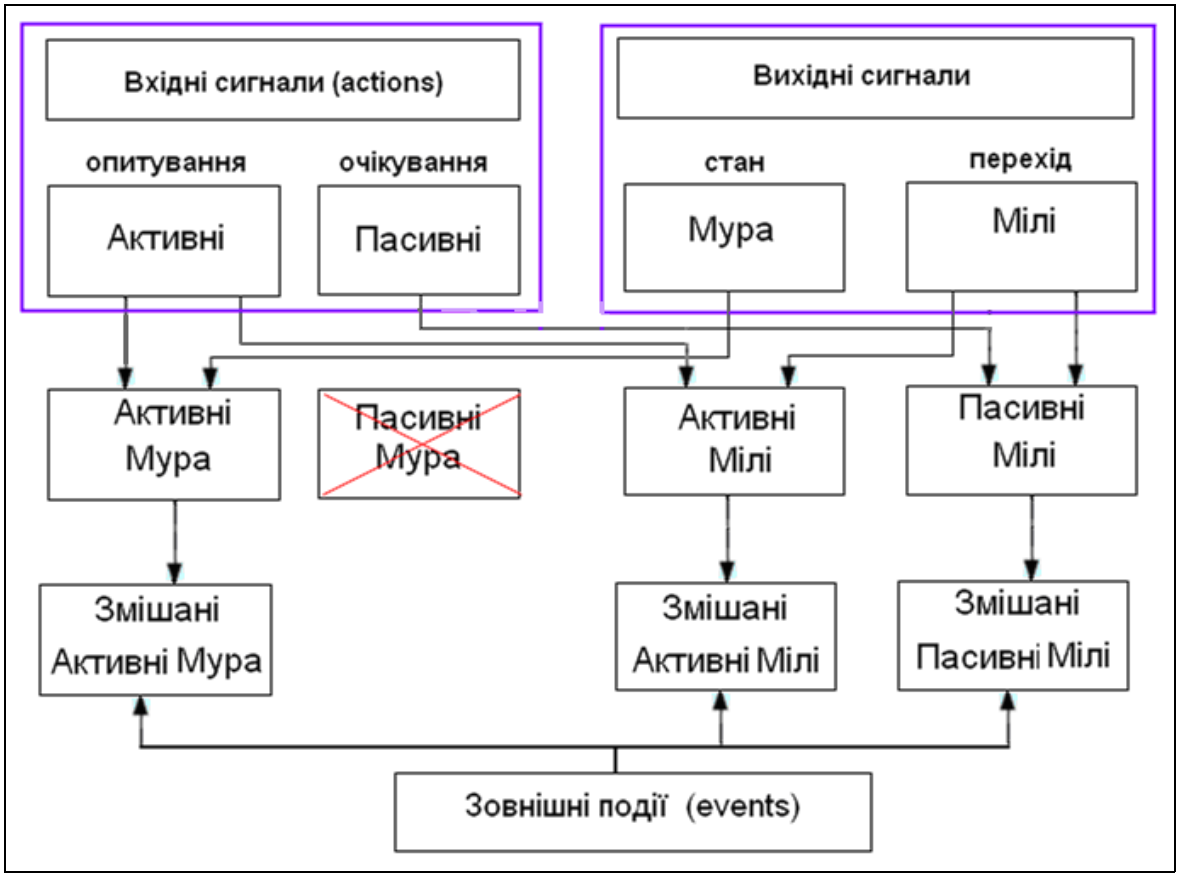


Рисунок 2.5 – Загальна класифікація часових автоматів логічного управління

3 РЕАЛІЗАЦІЯ МЕТОДІВ ПОБУДОВИ ПОДІЄВИХ HDL-МОДЕЛЕЙ КЕРУЮЧИХ АВТОМАТІВ

3.1 Подієві HDL-моделі автоматів Мура

Якщо розглядати автомати за класифікацією по вихідним сигналам, то автомати Мура, як правило, розглядаються як активні або змішані. Такі моделі складають більшу частину керуючих автоматів в системах логічного управління, тому що досить просто встановити відповідність між технічними станами керованого об'єкта і набором керуючих сигналів в станах керуючого автомата Мура. Пасивні автомати Мура розглядаються досить рідко.

Таймаут для активного автомата Мура відповідає затримці знаходження у певному технічному стані. Активний автомат Мура зчитує (опитує) вхідний сигнал в момент активного фронту першого синхроімпульса після закінчення часу очікування to_i для поточного стану і «миттєво» переходить в новий стан. Вихідні сигнали автомата Мура пов'язані з поточним станом автомата. Для кожного вихідного сигналу y_j визначається затримка його появи (зміни) від моменту переходу у поточний стан. Після закінчення часу очікування вихідний сигнал може обнулятися або зберігати своє значення в залежності від особливостей алгоритму роботи конкретного автомата Мура [9].

Для реалізації автоматної моделі Мура з єдиною часовою змінною на мові опису апаратури застосовується додатковий лічильник `count`, який використовується для підрахунку числа автоматних тактів, протягом яких автомат реалізує певні часові параметри. Коли часовий автомат виконує перехід у новий стан, значення `count`, скидається в 0. У двохпроцесному шаблоні на мові VHDL призначення нового стану і нового значення лічильника відбуваються в єдиному процесі, пов'язаному з синхронізацією і установкою автомата в початковий стан.

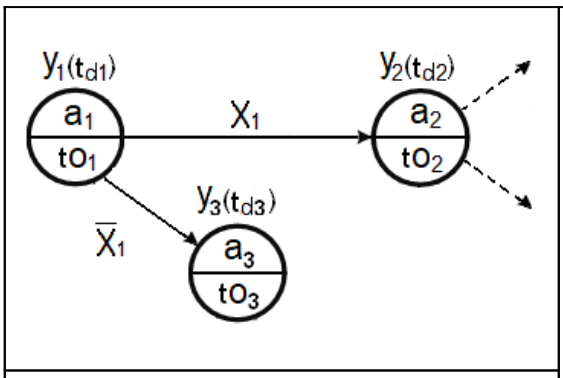
Таймаут to_i реалізується багаторазовим переходом зі стану в цей самий стан, при цьому число переходів визначається числом автоматних тактів таймаута. Значення лічильника порівнюється з (to_i-1), оскільки при переході у стан a_i автомат один такт знаходиться в ньому і, щоб таймаут був точно рівним to_i тактів, необхідно ще (to_i-1) тактів повторення. У темпоральном графі переходів таймаут станів вказуються всередині кіл позначення станів і реалізуються за допомогою петель, умовами для яких є перевірки значення сигналу лічильника автоматних тактів, але ці петлі на графі, як і сигнал Clk, не позначаються.

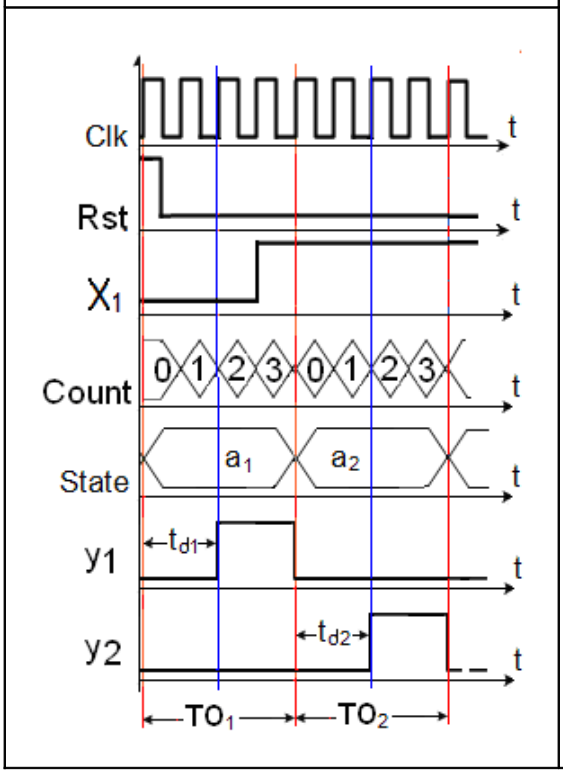
Затримки вихідних сигналів в поточному і-му стані t_{dij} визначаються в автоматних тактах від моменту переходу автомата у відповідний стан для кожного вихідного сигналу y_j і реалізується шляхом аналізу значень лічильника, який визначає затримку даного сигналу в автоматних тактах від моменту переходу у поточний стан. На темпоральном графі переходів вихідна затримка вказується в дужках поруч з кожним вихідним сигналом, а в VHDL-коді реалізується оператором умовного призначення сигналу поза процесом.

Модель активного автомата Мура при $to = 1$ і $t_d = 0$ збігається з моделлю традиційного мікропрограмного автомата.

На рис. 3.1 наведені фрагмент темпорального графа переходів часового автомата Мура, часова діаграма його функціонування і VHDL-код процесів синхронізації, призначення нового стану і нового значення лічильника.

Тут слід зауважити, що в другому процесі, в якому описується комбінаційна частина керуючого автомату, необхідно додавати оператор `next_count <= ( others => '0');` після стандартного `when others => next_state <= a1;` . Це важливо для коректного синтезу.






```
process (Clk, Rst)
  begin
    if Rst = '1' then   state <= a1;
          count <= ( others => '0' );
    elsif rising_edge(Clk) then
          state <= next_state;
          count <= next_count;
    end if;
end process;

process (state, count, x1)
  begin
  next_count <= (others => '0');
  case state is
    when a1 =>
      if count < TO1 - 1 then
          next_state <=a1;
          next_count <= count + 1;
      elsif x1 = '1' then next_state <= a2;
      else  next_state <= a3;
      end if;
   . . .
    when others => next_state <= a1;
                   next_count <= ( others => '0' );
  end case;
end process;

y1 <= '1'  when ( state = a1 and count >= td1 )
else '0';
y2 <= '1'  when ( state = a2 and count >= td2 )
else '0';
y3 <= '1'  when ( state = a3 and count >= td3 )
else '0';
```

Рисунок 3.1 – Специфікація і VHDL-модель часового автомата Мура

Для підтвердження достовірності отриманих результатів було виконано моделювання (behavioral) отриманих фрагментів опису VHDL-моделі часового автомата Мура з використанням інструментальних засобів CAD Xilinx ISE 14.7.

На рис. 3.2 наведені результати поведінкового моделювання часового автомата Мура для плати Spartan-3E Starter Kit з кристалом FPGA XC3S500E-5fg320. Результати моделювання у вигляді часових діаграм повністю співпали зі специфікацією, яка наведена на рис.3.1.

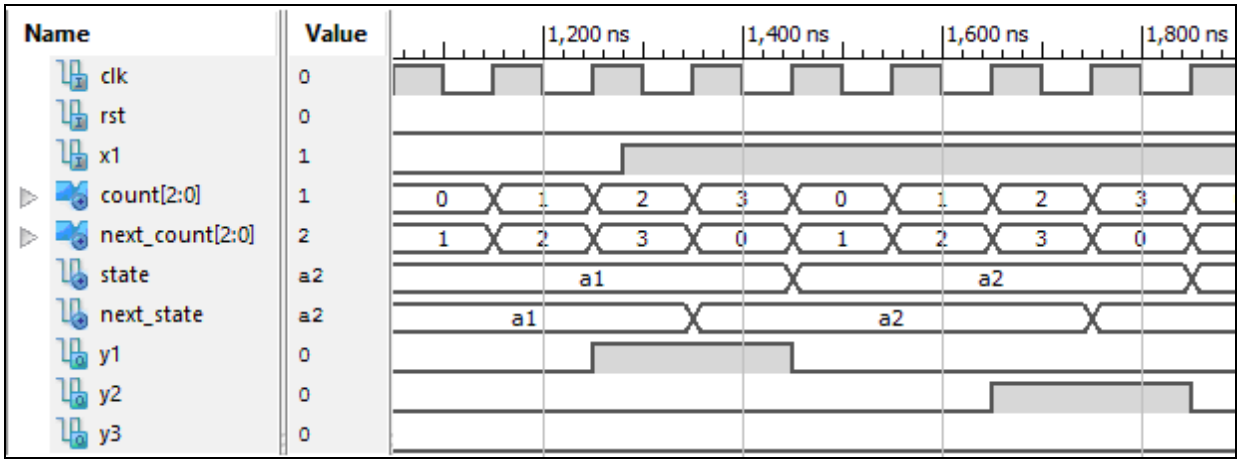


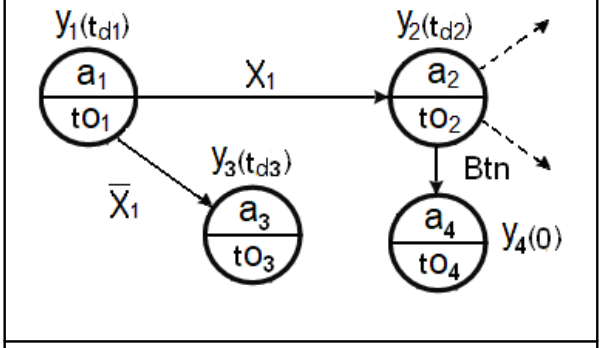
Рисунок 3.2 – Waveform моделювання роботи часового автомата Мура

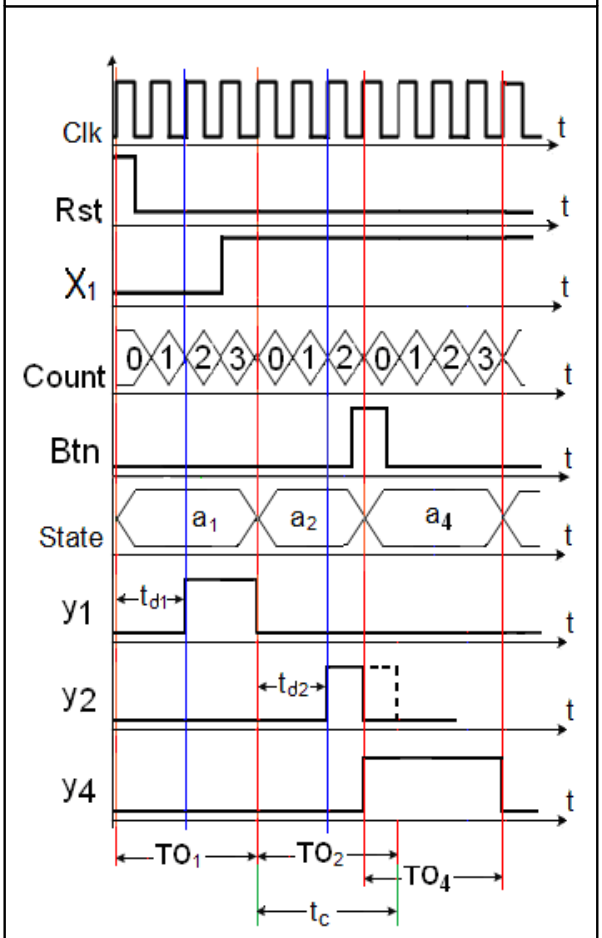
Якщо у активній моделі керуючого автомата Мура крім вхідних сигналів, що опитуються, використовуються зовнішні події Btn, які переривають знаходження у поточному стані, то таку модель прийнято називати змішаною. Зовнішня подія Btn перериває таймаут знаходження автомату у поточному стані та ініціює перехід до нового стану.

Часові обмеження t_c, протягом яких дозволена обробка сигналів зовнішніх подій в зазначеному стані автомата, визначаються як t_c = [cl, c2], де c1 - нижня межа часових обмежень в автоматних тактах, c2 - верхня межа зазначених обмежень. Для реалізації часових обмежень необхідно порівняння значення, сигналу лічильника, з нижньою та верхньою межами відповідного часового обмеження. Розглянемо нижню межу часового обмеження cl. Значення лічильника порівнюється з (cl – 1), оскільки при переході в стан a_i автомат один такт знаходиться в ньому до перевірки `count`, і, щоб нижня межа була точно cl тактів, необхідно ще (cl – 1) тактів перебування автомата в даному стані. Аналогічно, в останньому такті, в якому дозволена обробка зовнішньої події, значення лічильника порівнюється з (c2, - 1).

На рис. 3.3 наведені фрагмент темпорального графа переходів часового змішаного автомата Мура, часова діаграма його функціонування і VHDL-код процесів синхронізації, призначення нового стану, нового значення

лічильника та способу обробки події Btn.






```
process (Clk, Rst)
     begin
    if Rst = '1' then   state <= a1;
          count <= ( others => '0' );
    elsif rising_edge(Clk) then
          state <= next_state;
          count <= next_count;
    end if;
end process;
process (state, count, Btn, x1)
   begin
   next_count <= (others => '0');
   case state is
    when a1 =>
     if count < TO1 - 1 then
          next_state <=a1;
          next_count <= count + 1;
     elsif x1 = '1' then next_state <= a2;
     else  next_state <= a3;
     end if;
   when a2 =>
    if Btn = '1' then next_state <= a4;
    elsif count < TO2 - 1 then
          next_state <= a2.;
          next_count <= count + 1;
    else  next_state <= . . .;
    end if;
   . . .
when others =>
          next_state <= a1;
          next_count <= ( others => '0' );
  end case;
end process;
y1 <= '1'  when ( state = a1 and count >= td1 )
else '0';
y2 <= '1'  when ( state = a2 and count >= td2 )
else '0';
y3 <= '1'  when ( state = a3 and count >= td3 )
else '0';
Y4 <= '1'  when ( state = a4 ) else '0';
```

Рисунок 3.3 – Специфікація та VHDL-модель змішаного часового автомата Мура

На рис.3.3 представлені умови переходу по зовнішній події з урахуванням часових обмежень і часова діаграма процесу обробки зовнішньої події. Для спрощення викладення прийнято, що діапазон часових обмежень для a2 збігається з таймаутом даного стану $t_{c2} = [1, \infty]$, а затримка

появи вихідного сигналу для стану a4 дорівнює 0 ($td_4 = 0$). При цьому зовнішня подія спрацьовує на третьому автоматному такті у стані a2.

Для підтвердження достовірності отриманих результатів було виконано моделювання (behavioral) отриманих фрагментів опису VHDL-моделі змішаного часового автомата Мура з використанням інструментальних засобів CAD Xilinx ISE 14.7.

На рис. 3.4 наведені результати поведінкового моделювання часового змішаного автомата Мура для плати Spartan-3E Starter Kit з кристалом FPGA XC3S500E-5fg320. Результати моделювання у вигляді часових діаграм повністю співпали зі специфікацією.

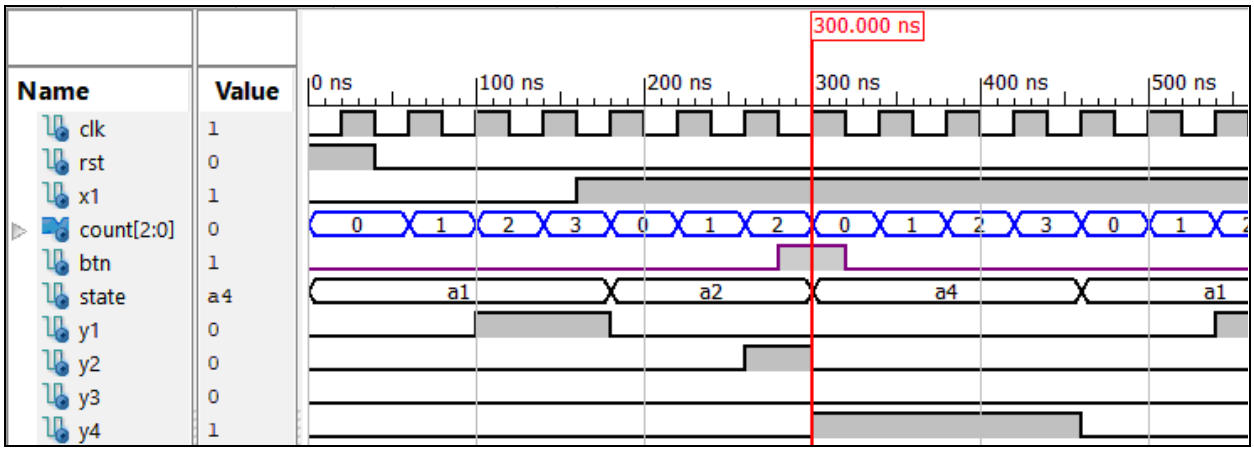


Рисунок 3.4 – Waveform моделювання (behavioral) моделі змішаного часового автомата Мура

На рис. 3.5 наведені результати моделювання часового автомата Мура після імплементації (Post-Place & Route Model Simulation). Результати моделювання співпали як з специфікацією так і з результатами поведінкового моделювання. Це підтверджує працездатність отриманої схемної реалізації змішаного подієвого часового автомата Мура.

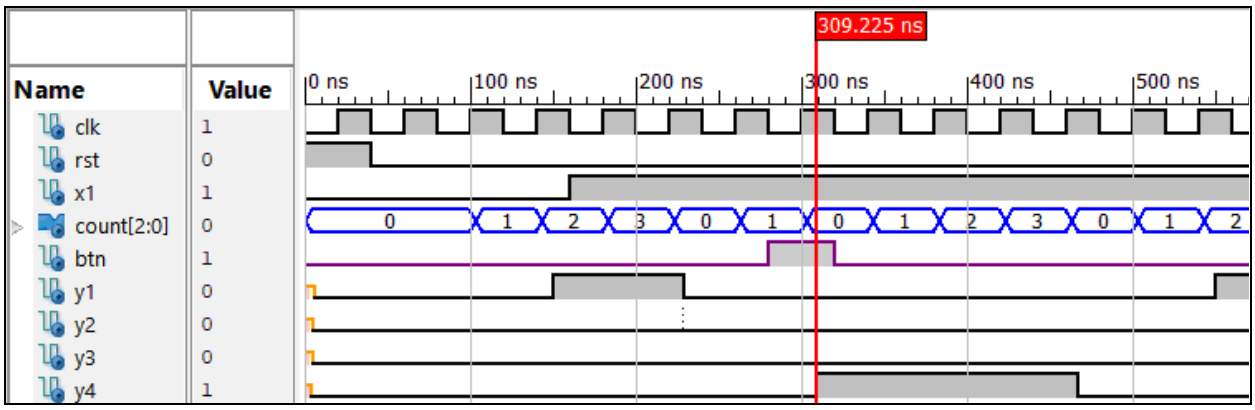


Рисунок 3.5 – Результати моделювання змішаного часового автомата Мура після імплементації (Post-Place & Route Model Simulation).

Повні результати моделювання та протоколи синтезу і імплементації для моделі змішаного автомата Мура наведені у додатку В.

3.2 Подієві HDL-моделі активних автоматів Мілі

Розглянемо різні варіанти моделей часових автоматів Мілі, які бувають як активні, так і пасивні в залежності від класу вирішуваних завдань.

Активний автомат Мілі зчитує (опитує) вхідний сигнал в момент активного фронту першого синхроімпульса після переходу автомата в поточний стан. Це ініціює початок переходу в наступний стан (a_i -> a_j), тривалість якого визначається таймаут to_i. До закінчення часу очікування автомат знаходиться в поточному стані. Вихідний сигнал y_j на переході з'являється з затримкою t_{dij} і триває до закінчення часу очікування. Після закінчення часу очікування вихідний сигнал y_j обнуляється і автомат переходить в новий стан. Таймаут і вихідна затримка реалізуються через аналіз лічильника автоматних тактів count.

Дана модель має дві особливості. По-перше, зміна вихідного сигналу залежить не від моменту появи вхідного сигналу, а «прив'язується» до найближчого робочого фронту синхроимпульса. Це обмеження пов'язане з особливістю побудови VHDL-моделей часових автоматів на основі аналізу

значення сигналу count.

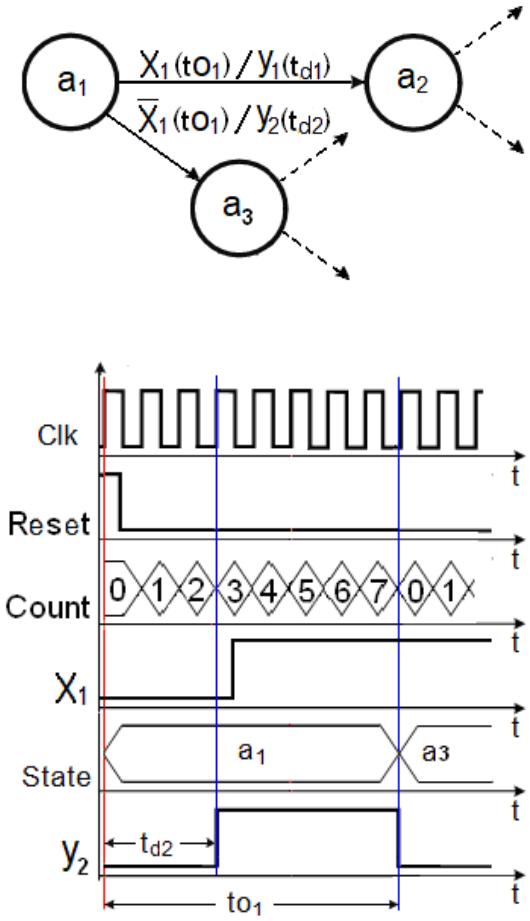
По-друге, активний автомат Мілі є по суті справи є змішаним, тому що вхідні сигнали розглядаються як опитувані вхідні дії, а крім цього в такої моделі можуть використовуватися події, які переривають перехід і, відповідно, вихідний сигнал.

Модель активного автомата Мілі досить близька до традиційного (мікропрограмного) автомату, а при to = 1 і $t_d = 0$ співпадає з нею.

Для реалізації моделі активного часового автомата Милі на мові опису апаратури VHDL пропонується виділити для кожного вхідного сигналу по сигналу-тригеру: signal x_stored. Тригери для вхідних сигналів x_stored запам'ятовують значення вхідного сигналу у момент першого фронту синхросигналу при вході в поточний стан. Це необхідно для блокуванні впливу зміни вхідного сигналу протягом переходу, але після першого фронту синхроімпульсу. Слід зазначити, що при синтезі схеми кожному такому сигналу відповідатиме синхронний тригер.

Таким чином, процес, що формує next_state і y, використовуючи стан автомата state і значення тригерів x_stored, може однозначно визначити поточний перехід. Відповідно до поточного переходу визначається таймаут, генеровані вихідні сигнали, а також відповідні вихідні затримки.

На рис 3.6 наведена специфікація активного часового автомата Мілі у вигляді фрагмента темпорального графа переходів і часова діаграма його функціонування, а також VHDL-код процесів синхронізації, призначення нового стану і нового значення лічильника. На цьому рисунку показана реалізація переходу a1 --> a3 з видачею вихідного сигналу y2 з затримкою t_{d2} (при X=0 в першому такті Clk і значення тригера x_stored при цьому буде "0"). При X = 1 в першому такті Clk значення тригера x_stored буде дорівнювати "1" (це визначає ініціалізацію переходу a1 --> a2 і видачу сигналу y1).




```
-- register input signals and events
process (Clk, Reset)
   begin
     if Reset = '1' then   x1_stored <= '0';
     elsif rising_edge(Clk) then
         if state /= next_state then
           x1_stored <= x1;                  -- x1='0'
         end if;
     end if;
end process;

combinational logic for next_state and Y
process ( state, x1_stored, count )
   begin
    y1 <= '0';    y2 <= '0';
    case State is
        when a1 =>
         if x1_stored = '1' then  --transition a1 —> a2
           if count < TO1 - 1 then next_state <= a1;
           else next_state <= a2;
           end if;
           if count >= Td1 then y1 <= '1';
           end if;
         else                          -- transition a1 —> a3
           if count < TO1 - 1 then next_state <= a1;
           else next_state <= a3;
           end if;
           if count >= Td2 then y2 <= '1';
           end if;
         end if;
. . .
     end case;
end process;
```

Рисунок 3.6 – Специфікація та модель активного автомата Мілі

На рис.3.7 наведені результати моделювання переходу a1 —> a3 для моделі активного автомата Мілі на рис.3.6.

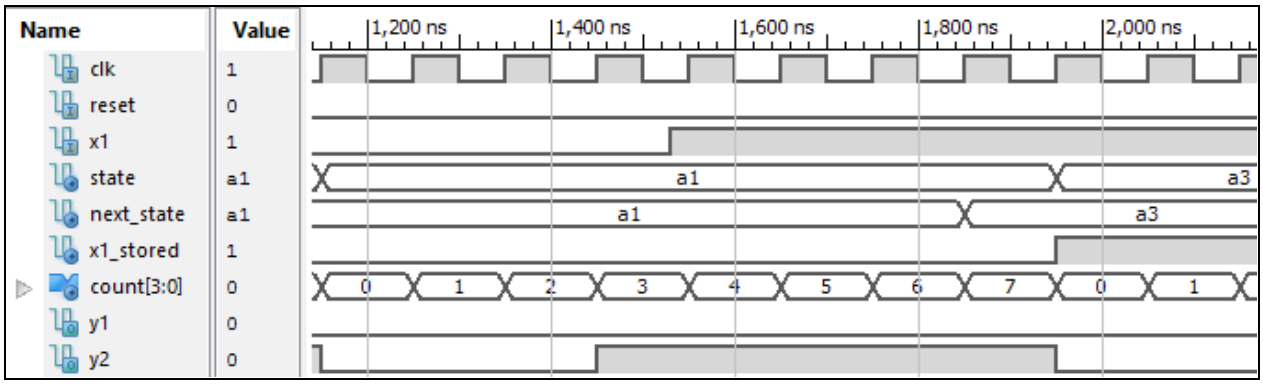


Рисунок 3.7 – Waveform моделювання роботи активного автомата Мілі

Якщо в активній моделі керуючого автомата Милі, окрім вхідних сигналів, що опитуються, використовуються перериваючі зовнішні події, то таку модель прийнято називати змішаною.

Представити часовий автомат Милі з перериваючою подією у рамках традиційної графової моделі не представляється можливим. Часові параметри, вихідні сигнали і обробка подій реалізуються під час переходів між станами; а переходам в графовій моделі відповідають дуги. Обробка перериваючих подій припускає, що для переходу e_j має бути ініційований новий перехід e_m, який є реакцією на перериваючу подію. Але традиційна графова модель не припускає наявності дуги, відповідної переходу e_m, оскільки дуги графової моделі не реалізують розгалуження.

Для часового автомата Милі з перериваючою подією пропонується використовувати модифікацію графової моделі. Модифікована графова модель змішаного часового автомата Милі - це дводольний граф з вершинами двох типів. Вершини першого типу відповідають станам автомата і зображуються у вигляді кіл. Вершини другого типу розміщуються на дугах, що виходять з вершин першого типу. Вершини другого типу реалізують обробку перериваючих подій і часові параметри, пов'язані з відповідним переходом, та зображуються як еліпс. Для вершини другого типу вказується стан автомата, з якого виконується перехід, часове обмеження (таймаут) і вихідна затримка для цього переходу. Якщо для цього переходу $t_{cij}=0$ або Btn = ∅ (відсутня перериваюча подія), то вершини другого типу можна не вказувати. Якщо даному переходу не відповідає жоден вихідний сигнал, то для цієї вершини $t_{dij} = \varnothing$ (рис. 3.8).

Факт наявності перериваючої події в інтервалі часових обмежень визначається шляхом аналізу значень лічильника автоматних тактів (TC0 – 1) <= count < TC1 і фіксується установкою в "1" сигналу-тригера Btn_meet_constraints. Тригер для перериваючих подій Btn_stored встановлюється в "1", якщо відбувається перша перериваюча подія, що

задовольняє часовим обмеженням у рамках таймауту цього стану, і скидається в "0" при переході між станами.

Таким чином, процес, що формує next_state та y, використовуючи стан автомата state і значення тригерів x_stored, Btn_stored може однозначно визначити поточний перехід. Відповідно до поточного переходу визначається таймаут, вихідні сигнали та відповідні їм вихідні затримки.

На рис 3.8 наведена специфікація активного автомата Мілі з перериваючою подією Btn у формі фрагмента темпорального графа переходів і часової діаграми його функціонування. На цьому рисунку показана наявність перериваючої події Btn на переході a1 -> a3 в інтервалі часових обмежень t_c та реалізація нового переходу a1 -> a2 з таймаутом to_b. При цьому сигнал y2 має нульову вихідну затримку.

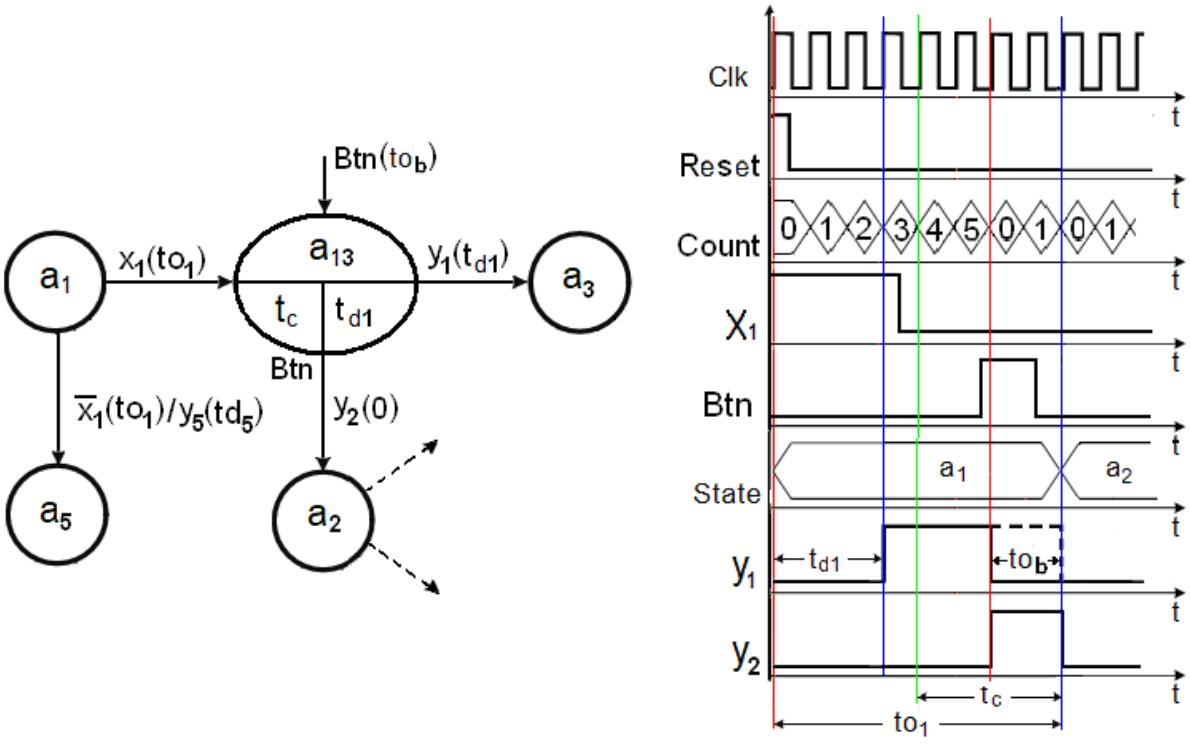


Рисунок 3.8 – Специфікація та часова діаграма активного автомата Мілі з перериваючою подією Btn

На рис 3.9 наведений фрагмент VHDL-моделі активного автомата Мілі з перериваючою подією, а саме код процесів фіксації перериваючої події, її попадання в інтервал часових обмежень, а також призначення нового стану і значень вихідних сигналів.

```
-- register input signals and events
process (Clk, Reset)
  begin
    if Reset = '1' then
          x1_stored <= '0';
          Btn_stored <= '0';
    elsif rising_edge(Clk) then
       if state /= next_state then
          x1_stored <= x1;
          Btn_stored <= '0';
       elsif Btn_meet_constraints = '1' then
          Btn_stored <= '1';
       end if;
    end if;
end process;

-- combination logic for Btn_meet_constraints
process ( state, x1_stored, Btn, count )
  begin
       Btn_meet_constraints <= '0';
       if Btn = '1' then
          if state = a1 and x1_stored = '1' and
           count >= TC0 - 1 and count < TC1
          then Btn_meet_constraints <= '1';
          end if;
       end if;
end process;
combinational logic for next_state and Y
process ( state, x1_stored, Btn_stored, count )
 begin
  y1 <= '0';   y2 <= '0';    y5 <= '0';
    case State is
      when a1 =>
       if x1_stored = '1' then
          if Btn_stored = '1' then
               if count < TOBtn – 1 then
                        next_state <= a1;
               else    next_state <= a2;
               end if;
              y2 <= '1';
          else
```

```
            if count < TO1 - 1 then
                next_state <= a1;
            else    next_state <= a3;
            end if;
            if count >= Td1 then y1 <= '1';
           end if;
         end if;
       else
           if count < TO1 - 1 then
                       next_state <= a1;
           else        next_state <= a5;
           end if;
           if count >= Td5 then y5 <= '1';
           end if;
       end if;
. . .
    end case;
end process;
```

Рисунок 3.9 – Фрагменти VHDL-моделі активного автомата Мілі з перериваючою подією Btn

На рис.3.10 наведені результати моделювання перериваючої події Btn на переході a1 —> a3 для VHDL-моделі активного автомата Мілі на рис.3.8.

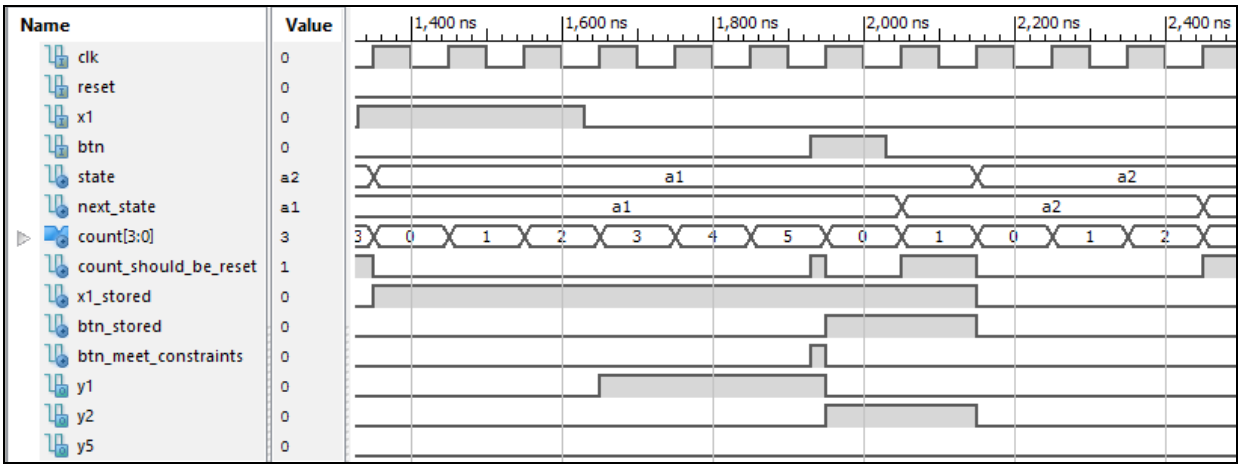


Рисунок 3.10 – Waveform моделювання роботи активного автомата Мілі з перериваючою подією Bt

3.3 Подієві HDL-моделі пасивних автоматів Мілі

Пасивний автомат Мілі в поточному стані очікує появи вхідного сигналу протягом проміжку часу, який визначається таймаутом to_i для даного i-го стану, і після приходу вхідного сигналу ініціює перехід новий стан з видачею вихідного сигналу тривалістю Δt. При цьому приймається, що $t_d = 0$. Якщо за період часу очікування вхідний сигнал не виникає, то автомат безумовно переходить у новий стан без видачі вихідного сигналу.

Особливістю пасивного автомата Мілі є нефіксований час T_i знаходження в i-му стані, яке складається з часу від початку таймаута до часу появи вхідного сигналу (події) t_e і часу тривалості вихідного сигналу Δt, тобто $T_i = t_e + \Delta t$. Пасивний автомат Мілі досить близький до традиційного (мікропрограмного) автомату, а при $to = 1$ і $t_d = 1$ збігається з ним.

На рисунку 3.11 наведена специфікація пасивного автомата Мілі в формі фрагмента темпорального графа переходів і часової діаграми його функціонування.

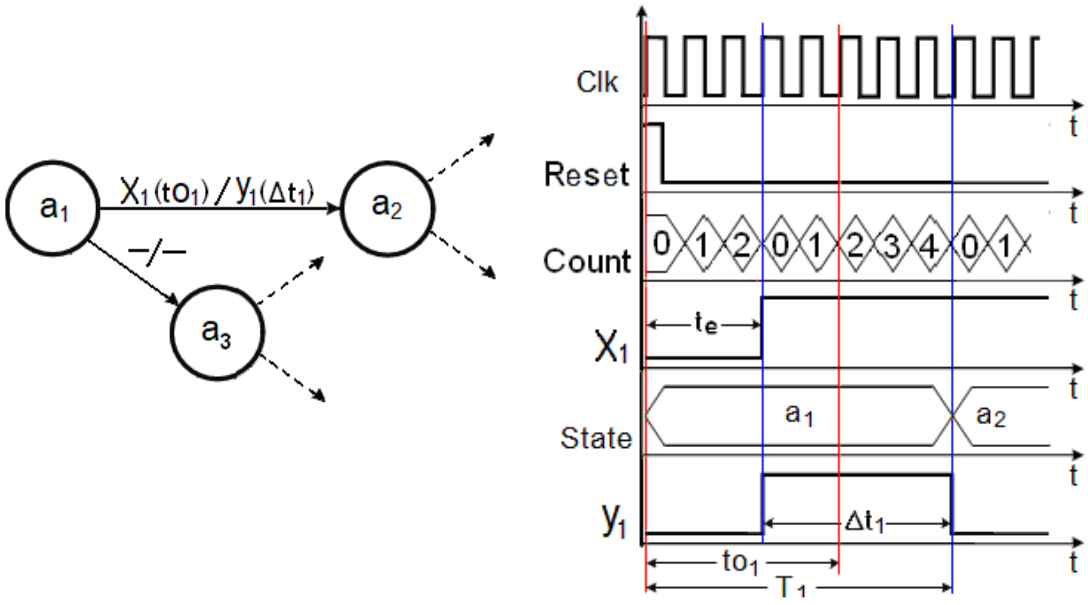


Рисунок 3.11 – Специфікація і часова діаграма пасивного (подієвого) автомату Мілі

При описі пасивного автомата Мілі кожному вхідному сигналу, події і переходу ставиться у відповідність по сигналу-триггеру: signal x_stored; signal Btn_stored, signal transition, signal next_transition. Сигнал transition (next_transition), визначає, чи виконується в даний момент перехід або автомат чекає події (зміни вхідного сигналу з 0 на 1 або навпаки) у відповідному стані.

Поки перехід не виконується, x_stored оновлюється кожен такт. Коли починається перехід, значення x_stored фіксується (на час переходу) в "1", якщо перехід - це реакція на зміну вхідного сигналу, в "0" - якщо перехід виконується після закінчення часу очікування (холостий такт). Значення x_stored на початку переходу може не відповідати значенню, яке потрібно зафіксувати тільки в такій (єдиній) ситуації: якщо при вході в стан і протягом всього часу очікування вхідний сигнал дорівнює "1". У такому випадку після закінчення часу очікування значення x_stored буде "1". З іншого боку, зміни вхідного сигналу "0" --> "1" протягом таймаута не відбувається, таким чином, повинен бути виконаний холостий такт, тобто потрібно зафіксувати "0" в x_stored. Для фіксації потрібного значення x_stored використовується сигнал x_stored_t.

Процес, який формує next_state, y, x_stored_t та next_transition, використовуючи стан автомата state і значення відповідних тригерів, визначає такі поточні ситуації:

- transition = "0", x_stored = "0" - автомат знаходиться в стані і чекає зміни вхідного сигналу "0" -> "1", якщо значення вхідного сигналу = "1", то починається перехід-реакція;

- transition = "0", x_stored = "1" - автомат знаходиться в стані і чекає зміни вхідного сигналу "0" -> "1", реакція на подію неможлива;

- transition = "1", x_stored = "0" - автомат виконує перехід (холостий такт) і в тригері x_stored фіксується значення сигналу x_stored_t після закінчення часу очікування;

- transition = "1", x_stored = "1" - автомат виконує перехід у відповідь на

зміну вхідного сигналу "0" -> "1", видаються вихідні сигнали відповідно до затримок, таймаут переходу визначено максимальною довжиною вихідного сигналу (Δt) .

Для автомата, що розглядається, процес, який формує next_state і y, використовуючи стан автомата state і значення тригера x_stored, може однозначно визначити поточний перехід. Значення "1" тригера x_stored говорить про те, що перехід в a2 був ініційований (подія X відбулося). Значення "0" тригера x_stored говорить про те, що події X в інтервалі to_1 не було, і автомат безумовно переходить до стану a3.

На рис. 3.12 наведено VHDL-код процесів установки автомата в початковий стан, фіксації ініціюючої події, запуск лічильника тактів в інтервал часу очікування, а також призначення нового стану і значення вихідних сигналів.

```
-- state synchronization
process (Clk, Reset)
  begin
   if Reset = '1' then state <= a1;
    elsif rising_edge(Clk) then state <= next_state;
   end if;
end process;
-- count synchronization
process (Clk, Reset)
   begin
     if Reset = '1' then  count <= (others => '0');
     elsif rising_edge(Clk) then
       if reset_count = '1' then count <= (others => '0');
       else count <= count + 1;
       end if;
     end if;
   end process;
-- transition synchronization
process ( Clk, Reset )
  begin
    if Reset = '1' then transition <= '0';
    elsif rising_edge(Clk) then
         transition <= next_transition;
    end if;
end process;
-- register input signals and events
```

```
process (Clk, Reset)
  begin
   if Reset = '1' then  x1_stored <= '0';
   elsif rising_edge(Clk) then
    if transition = '0' and next_transition = '1'
        then  x1_stored <= x1_stored_t;
    elsif next_transition = '0' then  x1_stored <= x1;
    end if;
   end if;
end process;
-- comb. logic for next_state, Y, next_transition
process ( state, x1_stored, x1, transition, count )
  begin
   y1 <= '0';
   x1_stored_t <= '0';
   next_transition <= '0';
      case State is
         when a1  =>
           if transition = '1' then
              if x1_stored = '1' then
                 if count < DLT1 - 1  then
                   next_state <= a1;
                   next_transition <= '1';
                 else   next_state <= a2;
                 end if;
                 if count < DLT1 then y1 <= '1';
                 end if;
              else next_state <= a3;
              end if;
           else                    -- timeout
              next_state <= a1;
              if x1_stored = '0' and x1 = '1' then
                 next_transition <= '1';
                 x1_stored_t <= '1';
            elsif count >= TO1 - 1 then
                 next_transition <= '1';
                 x1_stored_t <= '0';
            end if;
           end if;
      end case;
end process;
```

Рисунок 3.12 – Фрагменти VHDL-моделі пасивного автомата Мілі

На рис. 3.13 наведені результати моделювання переходу a1 —> a2 для моделі пасивного автомата Мілі на рис. 3.12.

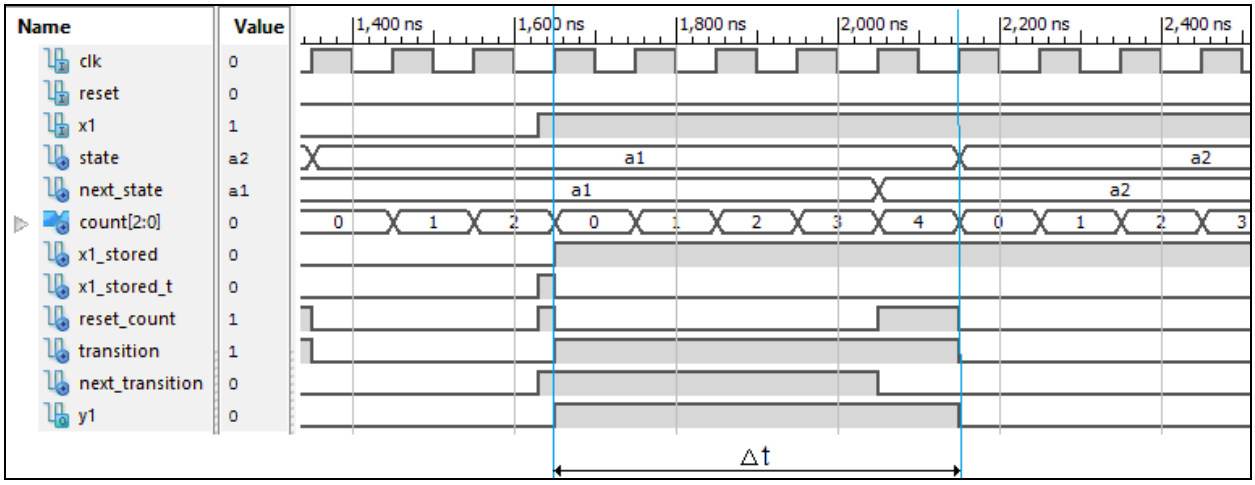


Рисунок 3.13 – Waveform моделювання роботи пасивного (подієвого) автомата Мілі

Якщо в моделі пасивного керуючого автомата Мілі, окрім вхідних сигналів, що що очікуються протягом таймауту, використовуються перериваючі зовнішні події, то таку модель прийнято називати змішаною.

Для пасивного часового автомата Мілі з перериваючою подією також пропонується використовувати модифікацію графової моделі у формі дводольного графу з вершинами двох типів.

На рисунку 3.14 наведена специфікація пасивного автомата Мілі з перериваючою подією Btn у формі фрагмента темпорального графа переходів і часової діаграми його функціонування. Для спрощення викладення прийнято, що діапазон часових обмежень для переходу (a1 --> a5) збігається з таймаутом даного переходу $t_{c1} = [1, \infty]$, а затримка появи вихідного сигналу для для переходу у стан a2 дорівнює 0 ($t_d = 0$) по визначенню.

Перериваюча подія Btn перериває перехід (a1 --> a5) таі ініціює перехід (a1 --> a2). Якщо ($t_b + \Delta t_2$) більше, ніж тривалість попереднього переходу T_1, то загальна тривалість переходу подовжується.

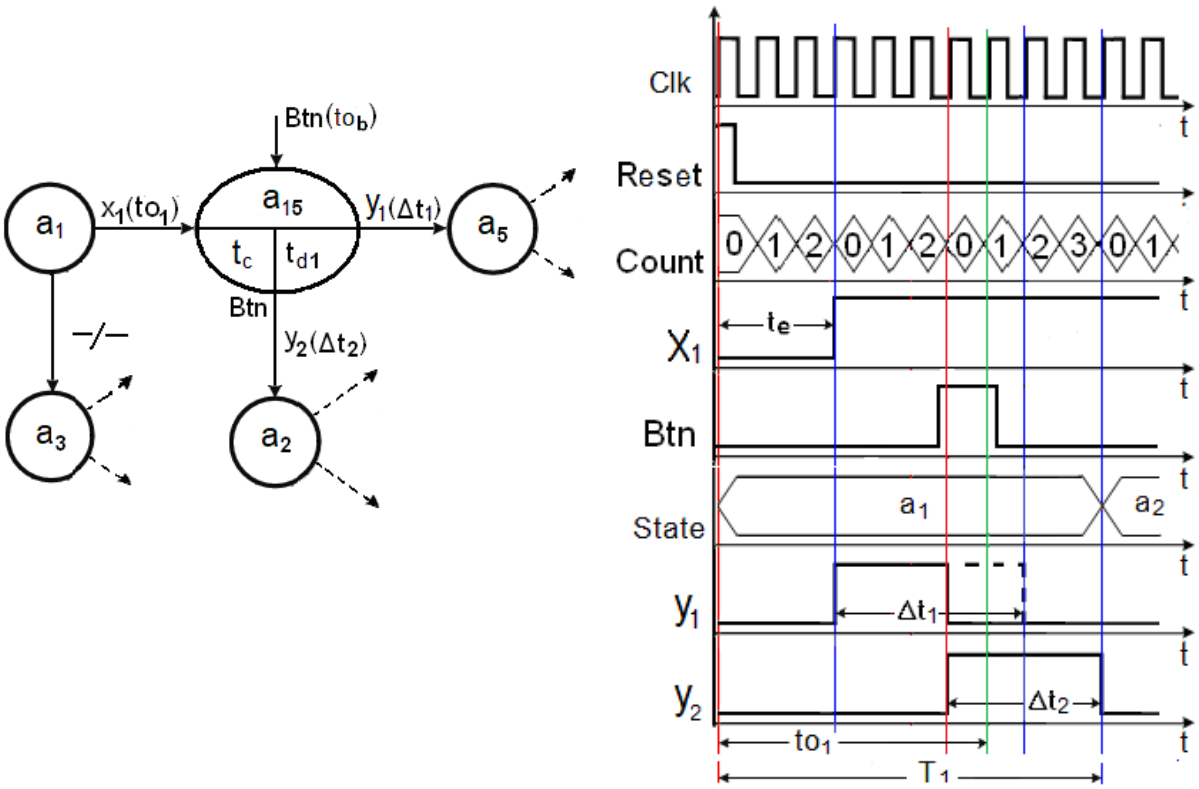


Рисунок 3.15 – Специфікація та часова діаграма пасивного автомата Мілі з перериваючою подією Btn

Тригер для перериваючої події Btn встановлюється в "1", якщо відбувається подія, яка задовольняє часовим обмеженням, і скидається в "0" при переході між станами. Таким чином, якщо Btn_stored = "1", то наразі відбувається перехід як реакція на подію Btn.

На рис. 3.15 наведено VHDL-код процесів установки автомата в початковий стан, фіксації перериваючої події, запуск лічильника тактів в інтервал часу очікування, а також призначення нового стану і значення вихідних сигналів.

```
-- combinational logic for Btn_meet_constraints
process ( state, x1_stored, Btn, count, transition )
   begin
      Btn_meet_constraints <= '0';
      if Btn = '1' then
         if state = a1 and x1_stored = '1' and transition = '1'
               and count >= constraint_L – 1 and count < constraint_H
         then Btn_meet_constraints <= '1';
         end if;
```

```
        end if;
end process;

 -- register input signals and events
 process (Clk, Reset)
  begin
    if Reset = '1' then
     x1_stored <= '0';  Btn_stored <= '0';
    elsif rising_edge(Clk) then
        if transition = '0' and next_transition = '1' then
                x1_stored <= x1_stored_t;
        elsif next_transition = '0' then
              x1_stored <= x1;  Btn_stored <= '0';
        end if;
        if Btn_meet_constraints = '1' then   Btn_stored <= '1';
        end if;
    end if;
end process;

process ( state, x1_stored, Btn_stored, x1, transition, count )
  begin
   y1 <= '0';
   x1_stored_t <= '0';
   next_transition <= '0';
      case State is
      when a1  =>
          if transition = '1' then
            if x1_stored = '1' then
             if Btn_stored = '1' then
                if count < output_delay_Y2 - 1 then
                   next_state <= a1; next_transition <= '1';
                else next_state <= a2;
                end if;
                if count < output_delay_Y2 then  Y2 <= '1';
                end if;
            else  next_state <= a3;              -- відповідає переходу без події
            end if;
          else
            next_state <= a1;
            if x1_stored = '0' and x1 = '1' then
                    next_transition <= '1';    x1_stored_t <= '1';
            elsif count >= T1 - 1 then
                    next_transition <= '1';    x1_stored_t <= '0';
            end if;
          end if;
```

Рисунок 3.15 – Фрагменти VHDL-моделі пасивного автомата Мілі з перериваючою подією

На рис.3.16 наведені результати моделювання перериваючої події Btn на переході a1 —> a5 для VHDL-моделі пасивного автомата Мілі.

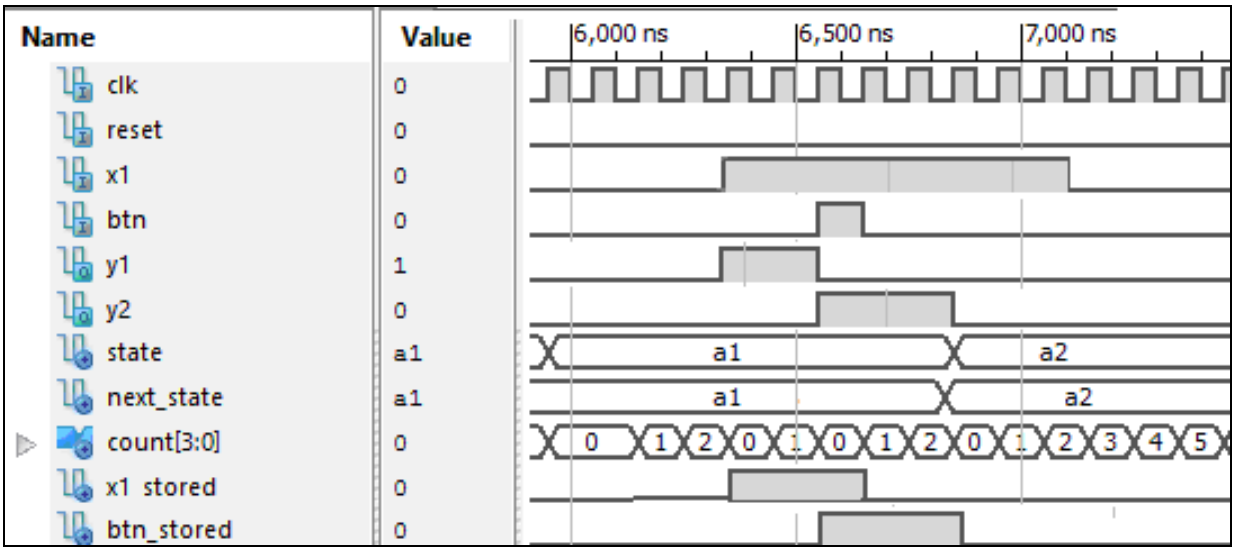


Рисунок 3.16 – Waveform моделювання роботи пасивного автомата Мілі з перериваючою подією Btn

3.4 Результати апаратної реалізації запропонованих автоматних HDL-моделей керуючих автоматів

Синтез і імплементація отриманих фрагментів VHDL-моделей на FPGA XC3S500E-5fg320 підтвердили їх коректність і працездатність. Результати часового моделювання після імплементації в цілому співпали з результатами поведінкового моделювання. У табл. 3.1 наведені результати синтезу та імплементації отриманих VHDL-моделей для всіх типів автоматів. Розрахункове число тригерів обчислювалося виходячи з кількості станів фрагмента моделі автомата, кількості додаткових керуючих тригерів і тривалості тайматута (що визначає кількість розрядів лічильника).

Таблиця 3.1 – Результати імплементації VHDL-моделей керуючих

автоматів в FPGA XC3S500E-5fg320

Тип автомата	Частота моделю-вания, MHz	Розра-хункове число тригерів	Flip-Flops	Latches	BELS	Slices/ LUTs
Мура активний	320,513	5	5	0	11	6/11
Мура активний з перивающою подією	256.430	5	5	0	21	9/18
Мілі активний	287,650	6	6	0	15	7/13
Мілі активний з перивающою подією	165,307	9	9	0	30	6/30
Мілі пасивний	232,626	7	7	0	15	8/15
Мілі пасивний з перивающою подією	141,898	11	11	0	42	9/45

На основі результатів, наведених у табл. 3.1, можна зробити висновок, що запропоновані моделі автоматів в цілому коректні, апаратні витрати не перевищують розрахункових параметрів, частотні характеристики знаходяться у прийнятних межах.

ВИСНОВКИ

Моделі часових керуючих автоматів в системах логічного управління, які реалізуються в hardware, характеризуються достатньою різноманітністю і мають досить багато класифікацій. За способом формування вихідних сигналів автомати підрозділяються на моделі Мура і Милі. За типом і способу обробки вхідних сигналів автомати класифікуються на опитувальні вхідні сигнали і подієві, а події в свою чергу класифікуються на зовнішні по відношенню до керованої системі і внутрішні, при цьому зовнішні події також мають багато класифікацій. За способом кодування станів автомати діляться на ряд класів, для яких апаратні витрати і швидкодія істотно розрізняються. За способом обробки часових параметрів для часових автоматів вони класифікуються на автомати з таймаут, часовими обмеженнями, часовими затримками і їх комбінаціями. За способом реалізації переходів і отримання вихідних сигналів FSM класифікуються на регулярні (regular), часові (timed) і рекурсивні, (recursive). Така різноманітність класифікацій керуючих автоматів в hardware обумовлено широкої областю їх застосування і різноманітністю вирішуваних завдань. Спосіб класифікації моделей FSM є визначальним при побудові шаблонів (patterns) HDL-моделей часових керуючих автоматів.

У даній роботі запропонована класифікація керуючих автоматів за способом отримання вихідних сигналів на моделі Мура і Милі, за способом обробки вхідних сигналів на активні і пасивні моделі і класифікація подій за способом їх обробки на ті, що ініціюють, та ті, що переривають. Дана класифікація дозволила побудувати VHDL-шаблони моделей часових керуючих автоматів для вирішення різноманітних завдань в системах логічного управління. Моделювання, синтез і імплементація в FPGA підтвердили приналежність розроблених шаблонів до підможини VHDL, що синтезується, та дотримання часових параметрів, заданих специфікаціями.

Реалізація запропонованих VHDL-моделей виконана на FPGA XC3S500E-5fg320. Результати діагностичного експерименту підтвердили коректність і працездатність. запропонованих моделей

Наукова новизна роботи полягає у подальшому розвитку методів побудови та апаратної реалізації HDL-моделей автоматних пристроїв логічного керування різних типів при використанні інструментальних засобів автоматизованого проектування фірми Xilinx.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Непейвода Н.Н. Стили и методы программирования. Курс лекций: учебное пособие / Н. Н. Непейвода. – М.: Интернет-университет информационных технологий, 2005. – 316 с.

2. Буч Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд.: пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.

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

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

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

6. Pedroni V. A. Finite state machines in hardware: theory and design (with VHDL and SystemVerilog) /Volnei A. Pedroni. – Cambridge, MA: MIT Press., 2013. – 338 p.

7. Shkil A. Design of real-time logic control system on FPGA / M. Miroschnyk, A. Shkil, E. Kulak, D. Rakhlis, I. Filippenko, M. Hoha, M. Malakhov, V. Serhiienko // Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS'19), Sept. 13-16, Batumi, Georgia, 2019. – P.488-491.

8. Baranov S. Logic and System Design of Digital Systems / S. Baranov. – Tallinn: TUT Press, 2008. – 267 p.

9. Shkil A. Hardware implementation of timed logical control FSM / M. Miroschnyk, A. Shkil, E. Kulak, D. Rakhlis, I. Filippenko, M. Malakhov, // Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS'20), Sept. 4-7, Varna, Bulgaria, 2020. – 6 p. [Електронний ресурс] / IEEE Xplore Digital Library – Режим доступу: www / URL: https://ieeexplore.ieee.org/document/9225129 – 20.10.2020 р. – Загол. з екрану.