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)

Моделі часових керуючих автоматів з гнучкою логікою на основі мікроконтролерів
(тема)

Виконав: студент 2 курсу, групи СКСм-20-1

_____Кислянська Д.В._____
(прізвище, ініціали)

Спеціальність *123_Комп'ютерна інженерія*

Тип програми ____*освітньо-професійна*_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

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

Керівник роботи доц. Шкіль О.С.__
(посада, прізвище, ініціали)

Допускається до захисту

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

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)

«__04__»___11____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Кислянській Дар`ї Вікторівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Моделі часових керуючих автоматів з гнучкою логікою на основі мікроконтролерів_____
_____
_____

Models of Timed Control Finite State Machines with Flexible Logic Based on Microcontrollers_____

затверджена наказом по університету від «_04_»__11__2021 р. № _1635 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____25.12.2021_____

3. Вихідні дані до роботи (проекту)_____

Мікроконтролер STM32F103T8T6_____

Мова програмування Сі_____

Середовище розробки Keil for ARM_____

Логічний аналізатор для апаратного налагодження Saleae Logic 8_____

_____

4. Перелік питань, що потрібно опрацювати у роботі _____

Автоматні моделі пристроїв керування _____

Моделі мікропрограмних автоматів з гнучкою логікою_____

Моделі часових автоматів Мура з гнучкою логікою_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 22 слайди

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання 02.09.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2021-08.09.2021	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2021-15.09. 2021	
3	Аналіз систем логічного управління	16.09.2021-29.09.2019	
4	Розробка автоматних моделей з гнучкою логікою	30.09. 2021-13.10.2021	
5	Опис автоматних моделей з гнучкою логікою для програмування для МК	14.10. 2021-31.10. 2021	
6	Програмування автоматних моделей з гнучкою логікою для МК	01.11. 2021-15.11. 2021	
7	Моделювання автоматних моделей з гнучкою логікою для МК	15.11. 2021-30.11. 2021	
8	Оформлення пояснювальної записки	01.12. 2021-15.12. 2021	
9	Захист проекту	15.12. 2021-25.12. 2021	

Студент _____ _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка містить 71 сторінку, 24 рисунки, 1 таблицю, 9 джерел за переліком посилань.

ЛОГІЧНЕ УПРАВЛІННЯ, ЧАСОВИЙ АВТОМАТ, СТАН, ГРАФ ПЕРЕХОДІВ, ГНУЧКА ЛОГІКА, МІКРОКОНТРОЛЕР.

Предметом дослідження в кваліфікаційній роботі є способи опису моделей керуючих автоматів з гнучкою логікою на основі мікроконтролерних пристроїв на мові програмування СІ. Проведений аналіз методів побудови керуючих мікропрограмних автоматів з гнучкою логікою, розроблений формат мікрокоманди часового автомата з гнучкою логікою, розроблена структура часового автомата з гнучкою логікою для реалізації в мікроконтролері.

Моделі часових автоматів представлені на мові програмування СІ для мікроконтролера STM32F103 у формі автоматного шаблону та у формі набору мікрокоманд з примусовою адресацією для гнучкого автомата. Виконано поведінкове моделювання запропонованих моделей, результати представлені у формі часових діаграм за допомогою програмного логічного аналізатора.

ABSTRACT

The explanatory note contains: 71 pages, 24 figure, 1 table, 9 sources according to the list of links.

LOGICAL CONTROL, TIMED FINITE STATE MACHINE, STATE, STATE DIAGRAM, PROGRAMMABLE LOGIC, MICROCONTROLLER.

The subject of research in qualification work is ways of describing models of the finite-state control machines with flexible logic based on microcontroller devices in the CI programming language. The analysis of methods for constructing microprogram finite-state control machines with flexible logic has been carried out, the format of the microcommand of a timed finite-state machine with flexible logic has been developed, the structure of a timed finite-state machine with flexible logic has been developed for implementation in a microcontroller.

Models timed FSM are presented in the CI programming language for the STM32F103 microcontroller in the form of FSM pattern and in the form of a set of microinstructions with forced addressing for a flexible automaton. Behavioral modeling of the proposed models has been carried out, the results are presented in the form of time diagrams using a logical software analyzer.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЛП– арифметико-логічний пристрій;

АЦП – аналого-цифровий перетворювач;

ГСА – граф-схеми алгоритмів;

ДП – дискретний пристрій;

ЗП –запам'ятовуючі пристрої ;

КА – керуючий автомат;

КАПЛ – керуючого автомата з програмованою логікою;

МК – мікроконтролер;

ОА – операційний автомат;

ОУ – об'єкт управління;

СЛУ – системи логічного управління

ПЗ – програмне забезпечення;

САПР – система автоматизованого проектування;

РЕП – радіоелектронні пристрої;

ФПВ – функції переходів-виходів;

ЦП – цифровий пристрій;

EEPROM – Electrically Erasable Programmable Read-Only Memory (енергонезалежна пам'ять даних);

IoT – Internet of Things (Інтернет речей);

NVRAM – Non Volatile Random Access Memory (енергонезалежна пам’ять);

RAM – Random Access Memory (оперативні запам'ятовуючі пристрої, ОЗП);

SRAM – Static Random Access Memory;

ROM – Read-Only Memory (постійний запам’ятовуючий пристрій, ПЗП);

SoC – System of Chip (Система на Кристалі).

ВСТУП

При проектуванні мікропрограмних систем управління складними технічними комплексами, що складаються з сукупності взаємодіючих блоків, можна йти двома шляхами. Один з них передбачає розглядати складні технічні комплекси як єдиний об'єкт і для управління цим об'єктом використовувати універсальну систему управління. альтернативний шлях орієнтований на побудову ієрархічної системи управління, де для кожного блоку комплексу розробляються свої системи управління, взаємодія яких підпорядковане одній загальній глобальної мети. Розробка і впровадження таких комплексів вимагають великих витрат часу і коштів.

Крім того, практично на всіх етапах життєвого циклу складних технічних комплексів з великою ймовірністю відбуваються спочатку непередбачений зміни. труднощі внесення змін як в проектовану систему в цілому, так і в алгоритми управління цією системою обмежують можливості її модернізації, розширення та спеціалізації сфери її застосування. Поставивши на чільне місце такі параметри складних технічних комплексів як адаптованість до можливих змін, економічність і швидкодія, можна припустити, що при побудові системи управління розглянутими об'єктами, найбільш ефективним є другий шлях. Його реалізація передбачає широке застосування керуючих автоматів з програмованою логікою. Даний підхід застосовується в розвиненій технології використання мікроконтролерів і ПЗУ в різних вбудованих системах.

Важливою частиною цифрових інформаційно-керуючих систем є пристрої управління, які можуть бути реалізовані у вигляді автомата з жорсткою або програмованої логікою. У більшості випадків, використовується модель автомата з жорсткою логікою, а саме - автомат Мура, що обумовлено відносною простотою і стійкістю, оскільки вихідні сигнали залежать тільки від станів автомата. Автомат Мура також забезпечує максимальну швидкодію за рахунок одночасного аналізу всіх логічних умов,

які служать вхідними сигналами схеми автомата.

Однак алгоритми функціонування систем управління мають структурну особливість, яка полягає в тому, що логічних вершин в такому алгоритмі досить багато - це відповідає частому опитуванням всіляких датчиків і інших зовнішніх пристроїв. Крім того, характерне чергування логічних і операторних вершин, що відповідає процесу «опитування датчика - реакція пристрої». Реалізація таких алгоритмів у вигляді керуючих автоматів з жорсткою логікою може виявитися неефективною з кількох причин: системи функцій переходів і виходів виходять громіздкими, терми мають велику розмірність, що неминуче веде до надмірного використання внутрішніх ресурсів пристрою при реалізації схеми. Крім того, ускладнюється автоматизація процесу формування систем функцій, що також призводить до помилок і ускладнює проектування і налагодження.

Автомати з програмованої (гнучкою) логікою (programmable logic finite state machine, PLFSM) дозволяють краще врахувати описані вище властивості алгоритмів управління об'єктами за рахунок явної адресації переходів. У процесі синтезу схеми автомата з програмованою логікою алгоритм управління перетворюється в масив бітових рядків, відповідних керуючим словами, і записується в керуючу пам'ять. Комбінаційна схема, яка забезпечує коректне зчитування і обробку керуючих слів, вимагає менше логічних ресурсів в порівнянні з автомата з жорсткою логікою, оскільки в моделі автомата з програмованою логікою реалізовані системи функцій мають набагато менший ранг і обсяг. Недоліком в цьому випадку є деяке зниження швидкодії з тієї причини, що логічні умови аналізуються в автоматах з гнучкою логікою по одному в кожному такті, і при великому числі логічні умови в алгоритмі управління зниження швидкодії буде істотним. Вирішити цю проблему в певній мірі можна шляхом використання ресурсів швидкодіючої вбудованої пам'яті [1].

1 МОДЕЛІ МІКРОПРОГРАМНИХ АВТОМАТІВ З ГНУЧКОЮ ЛОГІКОЮ

1.1 Автоматні моделі пристроїв керування в системах 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.

Пристрій логічного управління забезпечує реалізацію заданого алгоритму управління шляхом перетворення вхідних електричних сигналів, що надходять від датчиків та командних пристроїв, у вихідні управляючі впливи, що передаються на виконавчі пристрої. Пристрій логічного управління включає комплекс технічних засобів, що виконують функції логічного перетворення сигналів функції відліку часу, прямого та зворотного рахунку імпульсів, а також оперативної та жорсткої пам'яті.

У загальному випадку логічне управління включає наступні завдання.

1. Управління пуском/зупинкою об'єкта: визначається порядок включення в роботу, а також відключення при необхідності приводів основного та допоміжного обладнання, виконавчих механізмів, перетворювачів енергії апаратів тощо. Складається алгоритм управління пуском/зупинкою об'єкта та описується його робота.

2. Управління послідовністю технологічних операцій для об'єктів дискретної (циклічної) дії у функції часу та (або) залежно від виконання певних логічних умов. Складається алгоритм управління циклом роботи об'єкта, описуються операції циклу та умови переходу від операції до операції.

3. Управління обладнанням у передаварійних ситуаціях (захист від аварій), що виникають з різних причин, у тому числі і внаслідок хибних дій

персоналу. Складається алгоритм роботи системи автоматичного захисту та блокування на основі вимог, поданих у розділі 2, та описується його функціонування.

4. Керування світловими та звуковими приладами (сигналізація передпускова, про стан та положення елементів обладнання, про досягнення контрольованими параметрами заданих значень, про спрацювання захисту та блокування).

СЛУ впливають на об’єкт управління (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. Загальна структура СЛУ показана на рис. 1.

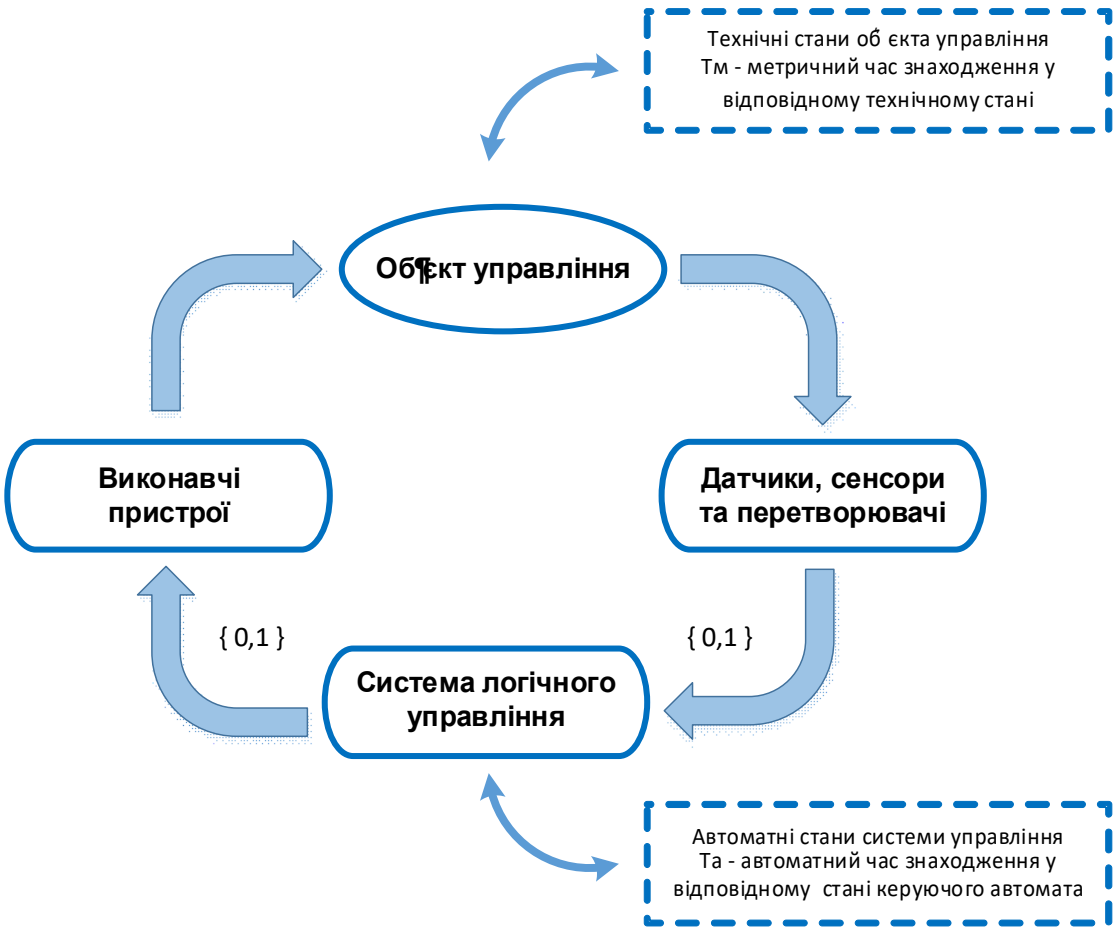


Рисунок 1.1 – Загальна структура системи автоматичного управління

В основі моделей систем логічного управління лежить поняття «стану». Стан це математична абстракція, яка однозначно ставиться у відповідність кожному з фізичних станів об'єкта управляння, так як зазвичай функціонування технічних систем проявляється через зміну їх станів. При цьому кожен стан в алгоритмі управління підтримує об'єкт управління в належному стані, а перехід в новий стан в алгоритмі призводить до переходу об'єкта в новий відповідний стан, що і забезпечує процес логічного управління об'єктом [2].

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.

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. Найбільш вживаною формою представлення абстрактного автомату є таблиця переходів-виходів (ТПВ) або граф переходів автомату (state diagram).

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. Зауважимо, 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.

При існуючому різноманітті вхідних форм опису проектів цифрових пристроїв можна виділити найбільш популярні у світі: аналітичні – мови опису апаратури, графічні або візуальні – ієрархічні цифрові структури і схеми, графи переходів автоматів. Як правило, кожна промислова САПР прагне мати усі можливі інтерфейси для того, щоб задовольнити найвибагливого споживача на ринку систем проектування.

Структурні моделі автоматів, використовувані в завданнях логічного управління, описуються зовсім в інших термінах. Символам вхідного та вихідного алфавітів ставляться у відповідність вектори вхідних та вихідних змінних, а при схемній реалізації – зовнішніх входів та виходів схеми. Структурний автомат функціонує у автоматному часі, який вимірюється в тактах {t, t+1, t+2 …}, тобто автомат переходить з одного стану до іншого за

один автоматний такт.

З відображенням станів абстрактного автомату на структурному рівні дещо складніше. Кожен стан абстрактного автомата a_i кодується вектором внутрішніх змінних z_i, і в аналітичному вигляді модель структурного автомату має вигляд $Y(t) = g(X(t), Z(t))$, $Z(t+1) = f(X(t), Z(t))$, де g – функція виходів структурного автомату, f – функція переходів структурного автомата. При цьому $Z(t+1) \equiv Z(t)$, але у наступному такті.

Таким чином загальноприйнята модель структурного автомата (модель Хаффмена), складається з комбінаційного і послідовністного компонентів. Послідовністний компонент містить елементи пам'яті, такі як синхронні тригери, які запам'ятовують стан і дозволяють змінювати його синхронно. Комбінаційний компонент складається з логічних елементів, які реалізують дві логічні функції: функцію виходів, яка обчислює значення вихідних сигналів, і функцію переходів, яка обчислює нові значення елементів пам'яті (тобто значення наступного стану).

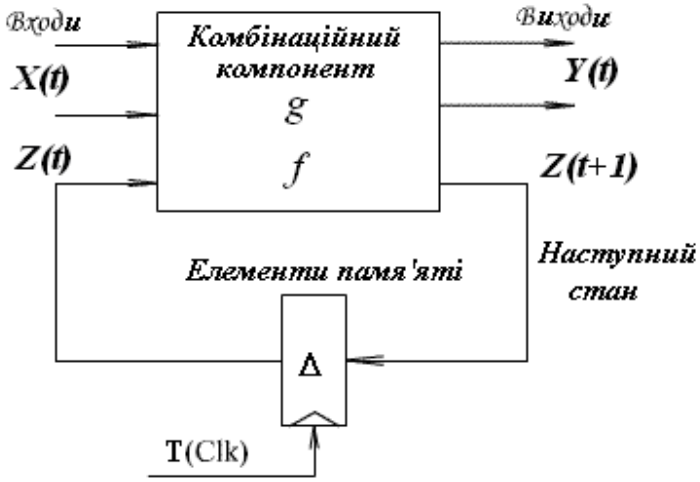


Рисунок 1.2 – Модель часового структурного автомату

Тривалість автоматного такту при цьому може бути одноковою для всіх стані автомату та залежати тільки від частоти синхропослідовності Clk,

мjже бути різною для кожного стану та мати тривалість Т.

Виходячи з цього для динамічних систем реального часу модель структурного автомату може бути розширена за рахунок введення у функцію переходів параметру автоматного часу, тобто, $Z(t+1) = f(X(t), Z(t), T)$,. а класичний граф переходів перетворюється у так званий темпоральний граф.

1.2 Модель мікропрограмного автомату з гнучкою логікою

Схема керуючого автомата може бути побудована на основі принципу програмного управління, що використовує операційно-адресну структуру слів, що управляють. При цьому під словами що управляють, розумітимемо набори сигналів мікрооперацій, що визначають порядок функціонування пристрою протягом одного такту. Такі набори (або списки дій) у завданнях проектування пристроїв керування отримали назву мікрокоманд. Відповідно безліч мікрокоманд, що реалізують заданий алгоритм перетворення даних, називають мікропрограмою.

У структурі автомата з програмованою логікою окремі елементи – мікрокоманди, що зберігаються у постійному запам’ятовуючому пристрої (ПЗП), виділяються у вигляді адреси. В окремому випадку адреса може бути обрана, наприклад, рівним номеру поточної мікрокоманди.

Управляюче слово у випадку має містити інформацію про мікрооперацiях, виконуваних у поточному такті, і навіть адресну інформацію про наступній мікрокоманді. Таким чином, структура слова, що управляє, достатня для подання мікрокоманди.

Реалізації мікропрограмного автомата на жорсткій логіці призводять до різних схемним рішенням в залежності від вихідних граф-схем алгоритмів (ГСА). А це означає, що при зміні функції автомата (зміні ГСА) необхідно змінювати або заново проектувати його схему. Цього недоліку позбавлені автомати з програмованою логікою. Алгоритм функціонування автомата з програмованою логікою представляється у вигляді закодованої прошивки,

яка може завантажуватися в перепрограмовані постійні запам'ятовуючі пристрої (ППЗП) або в оперативні запам’ятовуючі пристрої (ОЗП). Зміни в алгоритмі відображаються в мікропрограмі, але не впливають на схему автомата.

Зауважимо, що функція будь-якого керуючого автомата - генерування послідовності слів, що управляють (мікрокоманд), визначеної реалізованим алгоритмом з урахуванням значень інформаційних сигналів.

Якщо заздалегідь розмістити в запам'ятовуючому пристрої всі необхідні для реалізації заданого алгоритму (групи алгоритмів) мікрокоманди, а потім вибирати їх з пам'яті в порядку, передбаченому алгоритмом (з урахуванням значення інформаційних сигналів), то отримаємо автомат, структура якого слабо залежить від алгоритмів, що реалізуються, а поведінка в основному визначається вмістом запам’ятовуючого пристрою.

При змінах реалізованого алгоритму в досить широких межах структура такого автомата не змінюється, достатньо лише змінити вміст пам’яті запам’ятовуючого пристрою. Керуючий автомат, побудований за такими принципами, називається керуючим автоматом із гнучкою (програмованою) логікою.

Структурна схема такого автомата у найзагальнішому вигляді наведена на рис. 1.3. Автомат включає запам'ятовуючий пристрій мікрокоманд (зазвичай реалізується як ПЗУ), регістр мікрокоманд і пристрій формування адреси наступної мікрокоманди [3].

У кожному такті дискретного часу з пам'яті мікрокоманд зчитується одна мікрокоманда, що міститься в регістр мікрокоманд. Мікрокоманда містить два поля (дві групи полів), одне з яких визначає набір мікрооперацій, які в даному такті надходять до операційного автомата, а інше містить інформацію для визначення адреси наступної мікрокоманди.

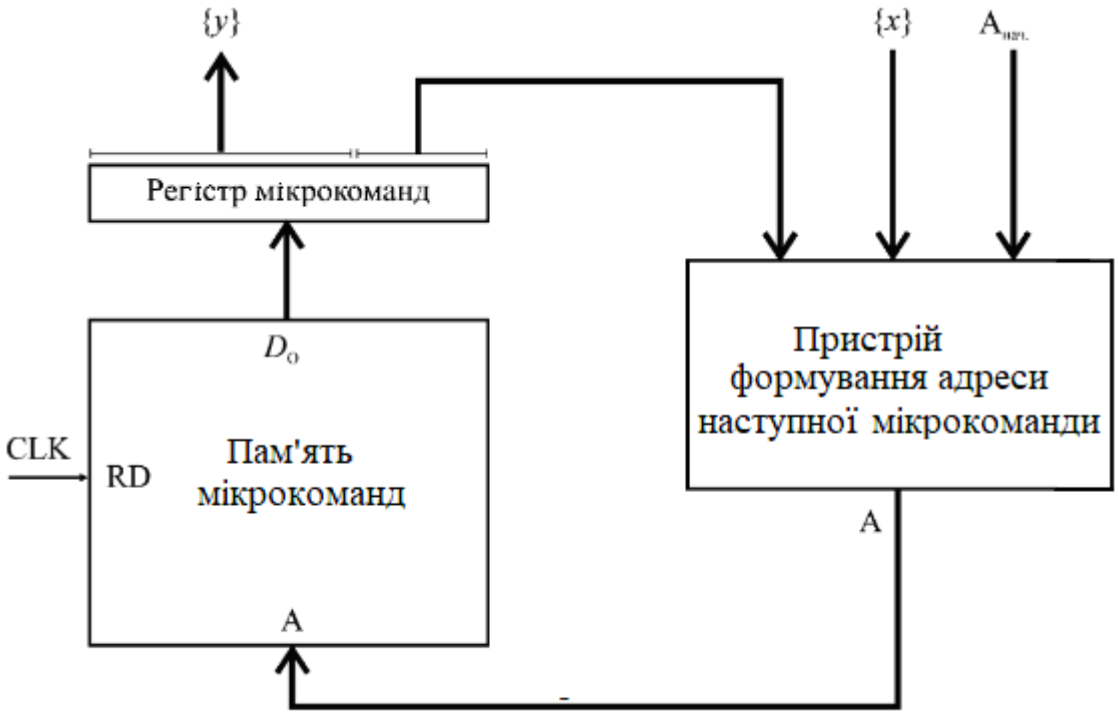


Рисунок 1.3 – Структура мікропрограмного автомата з гнучкою логікою

При проектуванні керуючого автомата з програмованою логікою (КАПЛ) необхідно вибрати формат (формати) мікрокоманд (мікрокоманди), способи кодування мікрооперацій та адресації мікрокоманд.

Розглянемо принципи управління по збереженій програмі для керуючих автоматів з гнучкою логікою. Функція керуючого автомата визначається:

– множиною вхідних сигналів (логічних умов): $X = \{x_1, x_2 \dots x_n\}$;

– множиною вихідних сигналів (мікрокоманд): $Y = \{y_1, y_2 \dots y_k\}$;

– мікропрограмою, яка задає порядок проходження вихідних сигналів Y залежно від значень вхідних сигналів X.

Найбільш наочним способом представлення функцій мікропрограмного керуючого автомата Мура є граф переходів і його таблична форма у вигляді прямої структурної таблиці переходів, які представлені на рис.1.4 [4].

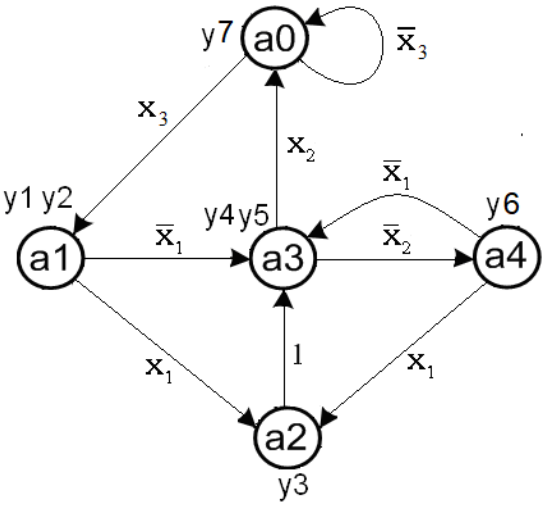
Якщо автомат Мура описаний прямою структурної таблиці переходів,

то для кожного стану a_m відомо:

– множина вихідних сигналів У (a_m), що виробляються автоматом в стані a_m;

– множина вхідних сигналів Х (a_m), що визначають всі можливі переходи автомата зі стану a_m в стан a_s;

– множина станів a_s, в які може перейти автомат зі стану a_m (позначимо А (a_m).



№	a_m	y_m	x_{ms}	a_s
0	a_0	y_7	$\overline{x_3}$ x_3	a_0 a_1
1	a_1	y_1y_2	x_1 $\overline{x_1}$	a_2 a_3
2	a_2	y_3	1	a_3
3	a_3	y_4y_5	x_2 $\overline{x_2}$	a_0 a_4
4	a_4	y_6	x_1 $\overline{x_1}$	a_2 a_3

Рисунок 1.4 – Графова та таблична моделі автомата Мура

Для кожного стану автомата Мура (а значить для кожної операторної вершини ГСА) цю інформацію, необхідну і достатню для опису закону функціонування автомата, можна закодувати і представити двійковим словом (мікрокомандою), яке має наступну структуру (рис. 1.5):

№	y_{m1}	y_{m2}	…	y_{mk}	x_{m1}	x_{m2}	…	x_{mn}	a_{m1}	a_{m2}	…	a_{ms}

Рисунок 1.5 – Структура мікрокоманди автомата з гнучкою логікою

де: № – номер двійкового слова, відповідного станом автомата a_m (або його адреса ЗП, де буде зберігатися мікропрограма); y_{mi} – вихідний сигнал (мікрокоманда), який повинен виробляти автомат в даному стані a_m; в загальному випадку таких сигналів k; ця частина двійкового слова називається операційної; x_{mi} – вхідний сигнал (логічне умова), яка бере участь хоча б в одному переході з a_m в a_s; в загальному випадку таких сигналів n; ця частина двійкового слова називається керуючої; a_{mi} – один зі станів a_s, в яке може перейти автомат зі стану a_m при певних значеннях логічних умов x_{mi} [5].

Оскільки опис одного стану - це одне двійкове слово, що має свою адресу в ЗУ, a_{mi} можна вважати адресою стану a_{mi} в ЗП; в загальному випадку таких станів s. Якщо кількість логічних умов в слові дорівнює n, то кількість a_{mi} одно 2^n. Ця частина двійкового слова називається адресною.

Якщо використовувати наведену вище структуру двійкового слова, то для опису даного автомата досить п'яти довічних слів (по числу станів автомата), що мають наступний формат:

№	y_{m1}	y_{m2}	x_m	$a_{x=0}$	$a_{x=1}$

Сам номер (№) в слово поміщати не потрібно, так як це адреса слова в ЗП. У слові під y_{mi} відведено два поля, так як максимальна кількість y_{mi} в стані a_m дорівнює двом; під x_{mi} - досить відвести одне поле, так як в будь-якому переході з a_m в a_{mi} (або a_s) бере участь тільки одне логічне умова. Під a_{mi} досить відвести 2 поля: для адреси переходу при $x_m = 0$ ($a_{x=0}$) і при $x_m = 1$ ($a_{x=1}$). Використовуючи обраний формат двійкового слова можна скласти у вигляді таблиці змістовну мікропрограму, що описує роботу автомата (рис. 1.4). У цій таблиці - кожен рядок відповідає одному з станів автомата (стовпець a_m); № - відповідає адресі слова, що описує цей стан в ЗП.

В колонках y_{m1} і y_{m2} записані мікрокоманд, що виробляються автоматом в стані a_m, в колонці x_m записані логічні умови, що визначають переходи зі стану a_m в стану, адреси яких в ЗУ записані в колонках $a_{x=0}$ і $a_{x=1}$, в колонці a_m записані стани автомата, яким відповідає даний рядок.

№	a_m	y_{m1}	y_{m2}	x_m	$a_{x=0}$	$a_{x=1}$
0	a_0	y_7	-	x_3	0	1
1	a_1	y_1	y_2	x_1	3	2
2	a_2	y_3	-	x_3	3	3
3	a_3	y_4	y_5	x_2	4	0
4	a_4	y_6	-	x_1	3	2

Рисунок 1.6 – Структура даних змістовної мікропрограми

Кожна з колонок цієї таблиці (крім a_m) має бути представлена двійковими кодами (рис. 1.7). Кількість двійкових розрядів для кожної колонки можна вибрати таким:

– y_{m1} – 3 розряди: тут будемо записувати номер y_{m1} (від 1 до 7);

– y_{m2} – 3 розряди: тут будемо записувати номер y_{m2} (від 1 до 7);

– x_m – 2 розряди: тут будемо записувати номер x_m (від 1 до 3);

– $a_{x=0}$ – 3 розряди: тут будемо записувати адресу $a_{x=0}$, по якому в автоматі має відбутися перехід до наступного слова при x_m =0 (від 0 до 4);

– $a_{x=1}$ – 3 розряди: тут будемо записувати адресу $a_{x=1}$, по якому в автоматі має відбутися перехід до наступного слова при x_m = 1 (від 0 до 4);

Таким чином, розрядність двійкового слова в нашому прикладі дорівнює 14. Закодована мікропрограма має вигляд, наведений в таблиці. У кожній колонці записано двійкові числа відповідно до прийнятих вище угодами. Необхідний обсяг пам'яті - 70 біт.

№	y_{m1}	y_{m2}	x_m	$a_{x=0}$	$a_{x=1}$
0	111	000	11	000	001
1	001	010	01	011	010
2	011	000	11	011	011
3	100	101	10	100	000
4	110	000	01	011	010

Рисунок 1.7 – Закодована структура даних змістовної мікропрограми

На рис.1.7 у рядку 2 описується стан автомата a_2. З цього стану - безумовний перехід автомата в стан a_3. Тому в стовпці x_m може бути записаний номер будь-якої змінної x_m, а в обох стовпчиках $a_{x=0}$ і $a_{x=1}$ однакову адресу переходу. Так як записаний в стовпці x_m обов'язково має одне зі значень: 0 або 1, то перехід буде безумовним за адресою $a_{x=0} = a_{x=1}$.

Схема, що забезпечує роботу з закодованої мікропрограмою, повинна містити наступні функціональні вузли (рис. 1.8):

- пам'ять для зберігання прошивки (ЗП) обсягом не менше п'яти слів; читання слова з пам'яті здійснюється за адресою, що подається на адресний вхід ЗП; розрядність слів - 14 біт;

- схему, перетворюючу двійкового коду y_{m1} і y_{m2} в унітарний код, відповідний закодованого y_i; ця схема - дешифратор (DCy);

- мультиплексор (MSX), що вибирає з множини вхідних сигналів X всього один, номер якого записаний в стовпці x_m;

- мультиплексор (MSa), що вибирає з слова адресу $a_{x=0}$ (якщо вибране мультиплексором MSX значення $x_{m=0}$) або адреса $a_{x=1}$ (якщо вибране мультиплексором MSX значення $x_m = 1$);

- регістр адреси (RG), в який необхідно по імпульсу синхронізації записати обрану мультиплексором MSa адресу і подати її на адресний вхід ЗП, з якого буде зчитане наступне слово (інакше - перехід в наступний стан).

y_{m1}			y_{m2}			x_m		$a_{x=0}$			$a_{x=1}$		
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Рисунок 1.8 – Структура мікрокоманди в запам’ятовуючому пристрої

Пронумеруємо розряди двійкового слова з 1 по 14 (як показано в

таблиці); номера розрядів будуть відповідати номерам виходів ЗУ. Функціональна схема автомата з гнучкою логікою приведена на рис. 1.9.

На цій схемі крім описаних вище елементів є 8 діз'юнкторов, які об'єднують однойменні виходи дешифраторів DCy_{m1} і DCy_{m2}. Ці елементи необхідні в разі, якщо одна і та ж мікрокоманда уі буде записана в різних рядках прошивки в різних стовпчиках y_{m1} і y_{m2}.

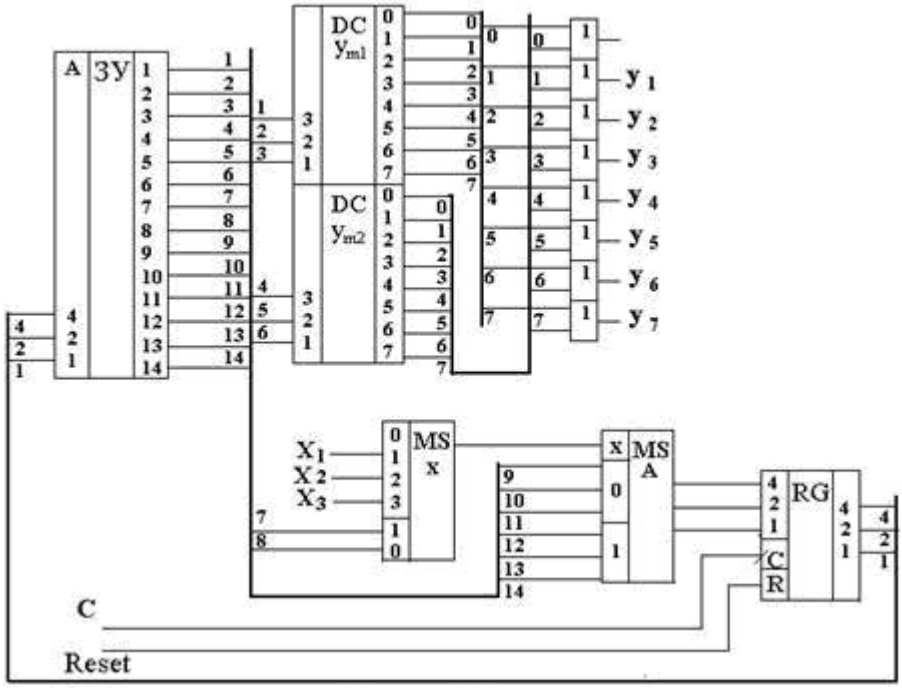


Рисунок 1.9 – Функціональна схема автомату з гнучкою логікою

Автомат (рис. 1.9) працює в такий спосіб.

Сигнал Reset = 1 встановлює значення «0» в усіх розрядах регістру RG. Код (або двійкове число) A = 000 з виходу RG подається на адресний вхід ЗУ. На виходах ЗП (розряди 1..14) з'являється слово, записане за адресою 0 (див. Таблицю). В цьому слові:

– розряди 1,2,3 - подаються на вхід DCy_{m1} і, оскільки в цих розрядах записано число 111 (7), на виході «7» DCy_{m1} формується 1, а значить і вихідний сигнал $У_7 = 1$;

– розряди 4,5,6 - подаються на вхід DCym2 і, оскільки в цих розрядах

записано число 000 (0), (яке ми не використовували для кодування y_i) на виході «0» DCy_{m1} формується «1», але ні на будь-яких інших виходах уі ця «1» чи не з'явиться;

– розряди 7,8 - подаються на керуючі входи мультиплексора MSX; в цих розрядах записано число 11 (3). Значення логічного умови x_3 з'являється на вихід MS_X;

– мультиплексор MSa, в залежності від значення x_3 («0» або «1») подає на входи регістра адреси RG або адреса переходу, записаний в розрядах 9,10,11 (якщо $x_3 = 0$), або адреса переходу, записаний в розрядах 12,13,14 (якщо $x_3 = 1$);

– під час вступу на вхід С імпульсу синхронізації, ця адреса запишеться в регістр RG і з його виходів надійде на адресний вхід ЗУ, на виході якого з'явиться слово, записане за цією адресою.

Таким чином, за один цикл (час між синхроімпульсами С) автомат зчитує з ЗП двійкове слово, виробляє мікрокоманду y_m, проаналізує вхідні сигнали x_m, і в залежності від їх значення вибирає адресу наступного двійкового слова в ЗП.

Спроектований автомат містить багато надлишковості і був приведений в якості прикладу, Недоліками такого підходу є, перш за все, складність схеми (при досить простих функціях автомата) і не оптимальна довжина мікрокоманд (двійкового слова), що призводить до неефективного використання пам'яті ЗП.

В автоматах з гнучкою логікою використовуються, в основному, два методи адресації мікрокоманд: примусова і природна.

При примусовій адресації адреса переходу до наступного бінарного слову в явному (або неявно) вигляді присутня в самій мікрокоманді. Так, в розглянутих вище прикладах, адресація була примусова. У двійковому слові для зберігання адрес переходів відводилося два поля (при одному логічному умови в слові). Така мікрокоманда називається двоадресний.

Природна адресація передбачає наявність довічних слів різного виду,

наприклад, операційні та керуючі слова. Операційні слова відповідають операційним вершин ГСА керуючого автомата, а керуючі - умовним. Якщо в ДСА автомата є лінійні частини (що не містять умовних вершин), то адреси слів, що описують такі друг за другом операторні вершини ГСА, повинні формуватися в природному наростаючому порядку: $a_{t+1} = a_t + 1$, де: a_t - адреса поточного слова, $a_t + 1$ - адреса наступного слова.

Керуюче слово може бути еквівалентно керуючий конструкції деякої мови програмування: IF $X_i = 1$ THEN GOTO $a_{xi=1}$; якщо значення логічного умови X і = 1, то адреса наступного слова $a_t + 1 = a_{xi=1}$; якщо $X_i = 0$, то адреса наступного слова: $a_{t+1} = a_t + 1$.

Формат мікрокоманд (довічних слів) при природній адресації може бути наступним. Операційна мікрокоманда містить два поля: поле ознаки виду мікрокоманд Р (операційна або керуюча; нехай для операційної - Р = 0) і поле, в якому будь-яким з можливих способів записані значення y_i, які повинен виробляти автомат. Формат операційного слова наведений нижче

0	$У_i$

Керуюча мікрокоманда містить три поля: поле ознаки Р (нехай для керуючої - Р = 1), поле логічного умови (може містити № логічного умови X_i) і поле адреси переходу при $X_i = 1$. Формат керуючого слова наведений нижче

1	X_i	Адр $a_{xi=1}$.

Оскільки для зберігання і операційних і керуючих слів використовується один і той же ЗУ, розрядність слів повинна бути однаковою.

На відміну від природної, примусова адресація зводиться до вказівкою в кожному слові адреси (або адрес) наступного слова. Однак навіть при двоадресний мікрокоманді, коли в кожному двійковому слові всього одне логічне умова, довжина адресної частини може бути значною. Є різні способи зменшення довжини адресній частині в довічним слові. Один з них полягає в наступному. В адресній частині слова записують адресу переходу

при Xi = 0 (Адр xi = 0), а якщо X i = 1, то адреса переходу Адр обчислюється так: $Адр = Адр_{xi=0} + 1$.

Якщо використовувати логічне значення змінної X_i як арифметичне (0 або 1), то адреса переходу можна обчислити так: $Адр = Адр_{xi=0} + X_i$.

Таким чином, скоротивши вдвічі довжину адресній частині довічного слова, ми не втратили можливість здійснювати умовний перехід в автоматі з природною адресацією. Однак при цьому отримали деяку незручність: виконавчі слова, до яких відбувається перехід, завжди знаходяться в ЗУ в сусідніх осередках, так як адреси у них різняться завжди на 1. Тому вбудованого автомата з описаним способом адресації може з'явитися надмірність у вигляді дублювання деяких слів і безумовних переходів.

Скористаємося прямою таблицею переходів автомата Мура з прикладу (рис. 1.4), і форматом мікрокоманд (рис. 1.5), в якому в адресній частині для природної адресації залишимо тільки одне поле $a_x = 0$ (рис. 1.10).

$К(У_i)$			x_m		$a_{x=0}$		
1	2	3	4	5	6	7	8

Рисунок 1.10 – Структура мікрокоманди з природною адресацією

Використовуємо кодування наборів мікрокоманд і додаткове ЗУ. Мікропрограма з коментарями наведена нижче.

$У_0 = \{y_7\}$; $У_1 = \{y_1, y_2\}$; $У_2 = \{y_3\}$; $У_3 = \{y_4, y_5\}$; $У_4 = \{y_6\}$;

$К(У_0) = 000$; $К(У_1) = 001$; $К(У_2) = 010$; $К(У_3) = 011$; $К(У_4) = 100$.

Для організації безумовного переходу в цьому прикладі введено додаткове логічне умова X0. У схемі автомата на вхід X0 подамо значення «0», а значить перехід в цьому випадку буде завжди за адресою $a_x = 0$, записаному в мікрокоманді.

Розглянуті вище схеми і формати мікрокоманд дозволяють аналізувати в одній мікрокоманді тільки одне логічне умова. Тому, якщо деякий перехід

залежить від декількох логічних умов, доводиться використовувати кілька мікрокоманд для опису цього переходу. Таким чином, якщо логічних умов в переході зі стану a_m в стан a_s два – то потрібно два такту роботи автомата (дві мікрокоманди), якщо три - то три такти тощо. Очевидно, що швидкодія автомата при цьому зменшується в два, три і більше разів.

1.3 Постановка задачі дослідження

Пристрої автоматного керування широко використовуються в сучасних системах логічного управління. Невисока ціна та простота налаштування мікроконтролерів залучає все більш широке коло користувачів до розробки власних пристроїв керування в системах логічного управління, особливо в галузі Internet of Things. Але більшість користувачів-початківців не володіє в достатній мірі програмуванням мікроконтолерів та способом використання автоматних шаблонів. Тому актуальною є проблема побудови мікроконтролерних пристроїв автоматного керування без програмування мікроконтролерів. Ця проблема може бути вирішена шляхом застосування керуючих автоматів з гнучкою логікою.

Таким чином, метою роботи є побудова моделі часового керуючого автомата з гнучкою логікою та реалізація цієї моделі в мікроконтролерних пристроях.

Об'єктом дослідження є моделі керуючих автоматів з гнучкою логікою в системах логічного управління.

Предметом дослідження є способи опису моделей керуючих автоматів з гнучкою логікою на основі мікроконтролерних пристроїв на мові програмування СІ.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз методів проектування систем логічного управління;

– провести аналіз методів побудови керуючих мікропрограмних автоматів з гнучкою логікою;

– розробити формат мікрокоманди часового автомата з гнучкою логікою;

– виконати програмну реалізацію та моделювання керуючого автомата з гнучкою логікою для мікроконтролера у запропонованій моделі;

– провести порівняння розробленої моделі мікроконтролерного автомата з гнучкою логікою з традиційною моделлю часового автомата на мові програмування СІ.

2 МОДЕЛІ ЧАСОВИХ АВТОМАТІВ МУРА З ГНУЧКОЮ ЛОГІКОЮ

2.1 Модель часового автомата

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 оповіщульних сигналів операційного автомата, а й зовнішніх по відношенню до керованої системі подій зовнішнього світу (external events), які для алгоритму управління є переривань.

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

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

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

При описі алгоритму функціонування цифрових пристроїв логічного керування в САПР цифрових пристроїв одним із стилів написання коду є стиль автоматного програмування. У автоматному програмуванні в якості базового використовується поняття «стан» [2]. 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). Часова змінна (timed variable) постійно збільшує своє значення і "скидається" в 0 при приході вхідного сигналу і перехід автомата в новий стан. Часові змінні вимірюються в автоматних тактах.

Для опису часових аспектів в автоматної моделі використовуються, як правило, три параметра: часові обмеження t_c (timing constraints), (вхідні)

таймаут t_{to} (timeouts) і вихідні затримки t_d (output delays), іноді називаються вихідними таймаут. Вхідний таймаут визначає максимальний час очікування вхідного впливу (події) для кожного стану автомата. Якщо вхідний символ ні поданий до закінчення часу очікування, то автомат починає опитування вхідних змінних і може перейти в інший стан. Часові обмеження є інтервалами на переходах, які обмежують час, протягом якого перехід може бути виконаний. Вихідні затримки (вихідні таймаут, вихідні затримки) відображають час, який витрачається автоматом на виконання переходу, тобто вихідний сигнал з'являться на виході через інтервал часу, який визначається вихідний затримкою [6].

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.

В залежності від мети та особливостей використання моделей часових автоматів існує безліч модифікацій таких моделей, які по різному враховують як спосіб обробки подій, так і спосіб врахування затримок в станах автомата. 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: $X = \left\{ X_C, X_E \right\}$ – множина вхідних змінних, X_C – множина оповіщувальних сигналів від об'єкта керування, X_E – множина зовнішніх подій; $Y = \left\{ Y_C, Y_F \right\}$ – множина

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

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

При переході автомата у поточній стан a_i для нього визначається основний часовий параметр $t_{to}(a_i)$ (timeout), тобто час протягом якого автомат має знаходитися у поточному стані, якщо зовнішня подія достроково не

переведе автомат у інший стан. t_{to} визначається у автоматних тактах. Після закінчення часу t_{to} автомат реагує на вхідні сигнали (опитує їх) та переходить у наступний стан.

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

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

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.

2.2 Модель часового автомату з гнучкою логікою

Як було зазначено у підрозділі 1.2, кожному рядку прямої структурної таблиці автомата Мура відповідає адресована мікрокоманда автомата з гнучкою логікою. На відміну від мікропрограмних автоматів у пряму структурну таблицю часового автомата Мура додаються стовпець таймаута to_i поточного станав (затримка у стані), стовпці затримок появи вихідних сигналів t_{dm} відносно початку таймауту та стовпець обробки зовнішніх подій (Btn). Для спрощення подальшого викладення приймемо, що t_c дорівнює таймауту.

Вихідними даними для проектування мікропрограмного КАПЛ є, як і для КА із «жорсткою» логікою, мікропрограма, представлена, як правило, у формі ГСА. Кожна операторна вершина має реалізуватися в один такт машинного часу, причому після операторної вершини в ГСА може йти □ операторна вершина, умовна вершина, обидва виходи якої або один з них з'єднані з операторними вершинами, ланцюжок із двох або більше умовних вершин, виходи яких з'єднані з умовними вершинами.

В свою чергу керуючі автомати в системах логічного управління представлені, як правило, темпоральними графами переходів. Замість умовних вершин ГСА розглядаються дуги, які виходять з поточної вершини графа переходів.

Граф переходів та рядок прямої структурної таблиці для одного стану часового автомата Мура представлений на рис. 2.1.

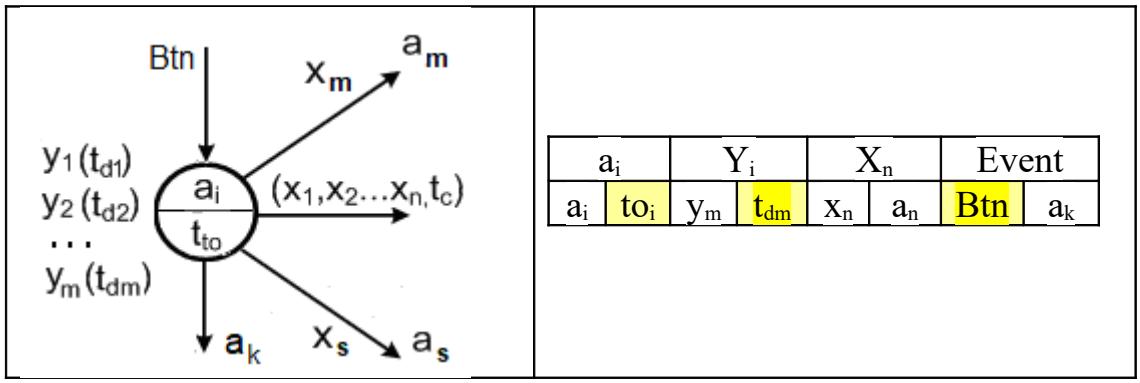


Рисунок 2.1 – Графова і таблична модель часового автомата Мура

Виходячи зі зміненої структури прямої структурної таблиці часового автомата Мура змінюється формат мікрокоманди часового автомата з гнучкою логікою, який представлений на рис. 2.2.

Розглянемо структуру мікрокоманди часового автомата з гнучкою логікою. В зазначеному форматі використовується примусова адресація. В перших чотирьох комірках вказується номер (адреса) стану, його таймаут у метричному часі, кількість активованих вихідних сигналів у цьому стані (Ny) та кількість можливих переходів (вихідних дуг) із даного стану.

a_i				Y_i		X_n		Event	
adr	to_i	N_y	N_x	y_m	t_{dm}	x_n	adr(a_n)	Btn	adr(a_k)

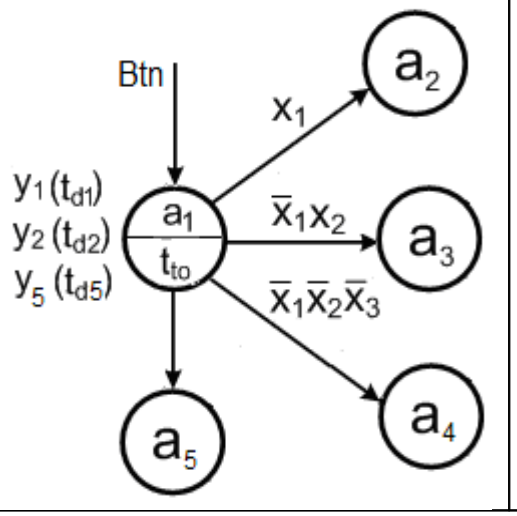
Рисунок 2.2 – Формат мікрокоманди часового автомата з гнучкою логікою

Для представлення вихідних сигналів будемо використовувати структуру списку, які активуються у поточному стані. На кожен вихідний сигнал відводиться дві комірки: номер вихідного сигналу та затримку появи вихідного сигналу у метричному часі відносно початку таймауту.

Для представлення вхідних сигналів будемо використовувати позиційну структуру всіх вхідних сигналів, які використовуються у даному автоматі. Кожній вихідній дузі графа переходів у поточному стані відповідає (m+1) комірок, де m – кількість вхідних сигналів (дій) для керуючого автомату, у останній комірці знаходиться адреса наступного стану. Позитивне значення вхідного сигналу x_i кодується 01, негативне значення $\overline{x_i}$ – 10, відсутність вхідного сигналу на даному переході – 00. В якості адреси наступного стану використовується адреса мікрокоманди відповідного стану у пам’яті мікрокоманд автомата з гнучкою логікою.

Останні дві комірки відведені для кодування події. Для спрощення наступного викладення припустимо, що в автоматі використовується одна подія. Кодування події наступне: 01 – реакція на подію присутня у поточному стані, 00 – подія у поточному стані не використовується.

Розглянемо структуру мікрокоманди керуючого автомата з гнучкою логікою на прикладі фрагмента графа переходів моделі часового автомата з зовнішньою подією, представленого на рис. 2.3. Структура мікрокоманди наведена на рис. 2.4.



to_1 - 5 sec; Ny=3;

t_{d1} – 2 sec; t_{d2} – 1 sec; t_{d5} – 0 sec;

01 – x_i; 10 – not(x_i);

00 – x_i не використовується.

Присутність події – 01,

відсутність події – 00.

Рисунок 2.3 – Фрагмент графа переходів часового автомата з подією

a_i				y_1		y_2		y_3	
a_1	5	3	3	y_1	2	y_2	1	y_5	0

$X(a_{12})$				$X(a_{13})$				$X(a_{14})$				Event	
x_1	-	-	a_2	$\sim x_1$	x_2	-	a_3	$\sim x_1$	$\sim x_2$	$\sim x_3$	a_4	Btn	a_5

Рисунок 2.4 – Структура мікрокоманди часового автомата з гнучкою логікою

Розглянемо кодування мікрокоманди автомата з гнучкою логікою. Для кодування адреси будемо використовувати чотирирозрядні коди, для кодування вихідних сигналів – трирозрядні коди, для вхідних сигналів – дворозрядні коди. Закодована мікрокоманда представлена на рис. 2.5.

0001	101	011	011	001	010	010	001	101	000

01	00	00	0010	10	01	00	0011	10	10	10	0100	01	0101

Рисунок 2.5 – Закодована мікрокоманда часового автомата з гнучкою логікою

3 ПОБУДОВА МОДЕЛІ АВТОМАТА З ГНУЧКОЮ ЛОГІКОЮ

3.1 Автоматна модель пристрою логічного керування

Реалізацію розроблених процедур верифікації пристроїв логічного керування розглянемо на прикладі пристрою керування тепловентилятором.

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

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

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

При цьому слід враховувати наступне. У відповідності до принципів

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

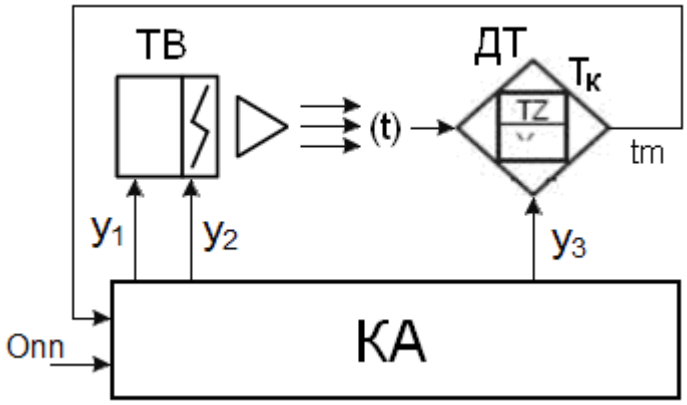


Рисунок 3.1 –Система автоматичного управління тепловентилятором

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

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

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

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

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

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

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

tm = {0, 1} – вихідний сигнал датчика температури, вхідний для

керуючого автомата (tm = 1 при $t > T_к$, tm = 0 при $t \leq T_к$, де t – температура теплого повітря, $T_к$, – критична температура теплого повітря, вище якої нагрів не здійснюється);

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

T_2=1 сек – проміжок часу між опитуваннями датчика температури при нагріві;

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

T_4=15 сек – проміжок часу для повного охолодження нагрівального елементу.

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

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

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

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

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

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

Для реалізації керуючого автомата будемо використовувати модель часового автомата Мура без вхідних обмежень та вихідних затримок, таймаути автомата Мура будуть реалізовані шляхом реалізації затримок у станах автомата Мура. Темпоральний граф переходів керуючого автомата наведений на рис. 3.2.

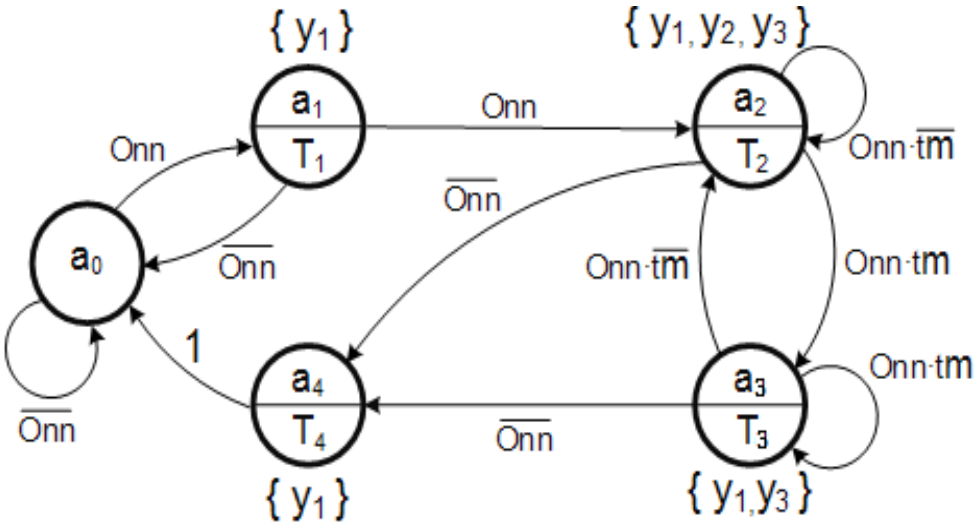


Рисунок 3.2 – Графова модель керуючого автомата

На основі графа переходів побудуємо пряму структурну таблицю автомата Мура з урахування стовпця затримок T_m для кожного поточного стану (рис. 3.3)

№	a_m	T_m	y_m	x_{ms}	a_s
0	a_0	-	-	$\overline{Onn}$ Onn	a_0 a_1
1	a_1	T_1	y_1	Onn $\overline{Onn}$	a_2 a_0
2	a_2	T_2	y_1 y_2 y_3	$Onn \cdot \overline{tm}$ $Onn \cdot tm$ $\overline{Onn}$	a_2 a_3 a_4
3	a_3	T_3	y_1 y_3	$Onn \cdot \overline{tm}$ $Onn \cdot tm$ $\overline{Onn}$	a_2 a_3 a_4
4	a_4	T_4	y_1	1	a_0

Рисунок 3.3 – Пряма структурна таблиця керуючого автомата

Виконаємо кодування станів та сигналів автомата у прямій структурній таблиці (рис. 3.4).

№	a_m	T_m	y_m	x_{ms}	a_s
0	a_0 000	-	-	$\overline{Onn}$ (10 00) Onn (01 00)	a_0 (000) a_1 (001)
1	a_1 001	T_1 0011	y_1 (01)	Onn (01 00) $\overline{Onn}$ (10 00)	a_2 (010) a_0 (000)
2	a_2 010	T_2 0001	y_1 (01) y_2 (10) y_3 (11)	$Onn \cdot \overline{tm}$ (01 10) Onn · tm (01 01) $\overline{Onn}$ (10 00)	a_2 (010) a_3 (011) a_4 (100)
3	a_3 011	T_3 0101	y_1 (01) y_3 (11)	$Onn \cdot \overline{tm}$ (01 10) Onn · tm (01 01) $\overline{Onn}$ (10 00)	a_2 (010) a_3 (011) a_4 (100)
4	a_4 100	T_4 1111	y_1 (01)	1 (00 00)	a_0 (000)

Рисунок 3.4 – Закодована пряма структурна таблиця часового керуючого автомата

Виходячи з того, що наведена модель часового автомата не має режимів обробки зовнішніх подій, структура мікрокоманди, яка наведена на рис. 2.2 дещо спрощується та представлена на рис. 3.5..

$adr(a_i)$				Y_i		X_n	
adr	to_i	N_y	N_x	y_m	t_{dm}	x_n	$adr(a_n)$

Рисунок 3.5 – Формат мікрокоманди часового автомата з гнучкою логікою без обробки подій

Виходячи зі структури мікрокоманди, наведеної на рис. 3.5, представимо закодовані мікрокоманди, які відображають логіку роботи автомата з гнучкою логікою. Відмітимо, що у наведеній структурі використовується примусова адресація, і адреса кожної мікрокоманди співпадає з кодом стана автомата при послідовному способу кодування станів.

a_1				Onn tm | a_n			
000	0000	000	010	10 00	000	01 00	001

a_1				y_1	Onn tm | a_n			
001	0011	001	010	0100	01 00	010	10 00	000

a_2				y_1	y_2	y_3	Onn tm | a_n					
010	0001	011	011	0100	10 00	11 00	01 10	010	01 01	011	01 10	100

a_3				y_1	y_3	onn tm | a_n					
011	0101	010	011	0100	11 00	01 10	010	01 01	011	01 10	100

a_4				y_1	onn tm | a_n	
100	1111	001	001	0100	00 00	000

Рисунок 3.6 – Структура мікрокоманд часового автомата з гнучкою логікою

3.2 Реалізація автомата з гнучкою логікою в мікроконтролері

Мікроконтролери, що використовуються у різних пристроях, виконують функції інтерпретації даних, що надходять із клавіатури користувача або від датчиків, що визначають параметри довкілля, забезпечують зв'язок між різними пристроями системи та передають дані іншим приладам. Застосування мікроконтролерів дозволяє значно знизити кількість та вартість використовуваних матеріалів та комплектуючих виробів, що забезпечить зниження собівартості кінцевої продукції. Використання мікроконтролерів може суттєво збільшити привабливість продукції для споживача завдяки реалізації «дружнього інтерфейсу» за відносно невеликих додаткових витрат. Забезпечується також можливість розширення галузі застосування продукції, що випускається, шляхом використання тих самих апаратних засобів з різноманітним програмним забезпеченням, спеціалізованим для реалізації різних функцій. При розробці систем управління різними процесами та об'єктами використання мікроконтролерів дає проектувальнику значні переваги.

При реалізації автоматів з гнучкою логікою важливо враховувати особливості побудови пам’яті мікроконтролерів. Більшість сучасних мікроконтролерів мають Гарвардську архітектуру та містять 3 види пам'яті:

– пам'ять програм FLASH;
– оперативна пам'ять (ОЗП) SRAM (Static RAM);
– енергонезалежна пам'ять даних EEPROM.

Адресні простори зазначених видів пам'яті зазвичай розділені. Способи адресації та доступу до цих областей пам'яті також різні. Така структура дозволяє центральному процесору працювати одночасно з пам'яттю програм, і з пам'яттю даних, що значно збільшує продуктивність. Кожна з областей пам'яті даних (SRAM та EEPROM) також розташована у своєму адресному просторі.

Пам'ять програм є такою, що електрично стирається, ППЗП (FLASH) і може підтримувати команди з розрядністю більше 8 біт. У деяких мікроконтролерах пам'ять програм розділена на 2 секції:

– секцію завантажувача (Boot Program);
– секцію прикладних програм (Application Program).

Пам'ять програм найчастіше є електрично перепрограмується, кількість циклів перезапису перевищує 10 тисяч. Більшість мікроконтролерів підтримують внутрішньосхемне програмування, тобто. завантаження програми в мікроконтролер можна здійснювати після монтажу на плату за допомогою спеціального роз'єму програмування. Для адресації пам'яті програм використовують лічильник команд (Program Counter – PC).

У пам'яті програм також знаходиться вектор скидання - в момент подачі живлення мікроконтролер починає виконання програми з цієї адреси, і тут розміщується команда переходу до програми, що виконується. Крім того, пам'ять програм містить таблицю векторів переривань. У разі переривання після збереження в стеку поточного значення лічильника команд відбувається виконання команди, розташованої за адресою відповідного вектора. Тому за даними адресами розміщуються команди переходу до

підпрограм обробки переривань. Положення вектора скидання та таблиці векторів переривань може бути перенесено з секції прикладних програм до секції завантажувача.

У деяких випадках пам'ять програм може використовуватися не тільки для зберігання програмного коду, але і для зберігання різних констант.

Оперативна пам'ять, як правило, містить 3 області:

– регістри загального призначення;
– службові регістри;
– пам'ять для зберігання даних.

Регістри загального призначення (РЗП) знаходяться у безпосередній близькості до АЛП. Однак у мікроконтролерах деяких фірм (зокрема, PIC фірми Microchip) є лише один робочий регістр, що грає роль одного з операндів у командах. Застосування набору регістрів загального призначення у поєднанні з конвеєрною обробкою дозволяє АЛП виконувати одну операцію (витяг операндів з набору регістрів, виконання команди та запис результату назад у регістр) за один такт.

Службові регістри мають свої ім'я, адресу та призначення. Вони призначені для конфігурації та обслуговування периферійних вузлів мікроконтролера. Коротка характеристика службових регістрів має бути наведена у посібнику з використання мікроконтролера (Data Sheet). Серед службових регістрів є, як правило, один регістр, який найчастіше використовується в процесі виконання програм. Це регістр стану. Він містить набір прапорів, які показують поточний стан мікроконтролера. Більшість прапорів автоматично встановлюються в "1" або скидаються в "0" при настанні певних подій (відповідно до результату виконання команд). Усі біти цього регістру доступні як читання, так запису. Ця інформація аналізується під час виконання умовних переходів. У разі переривань вміст регістру стану необхідно зберігати програмно (найчастіше це «піклуванням» компілятора). Решта оперативної пам'яті призначена для зберігання даних користувача.

Енергонезалежна пам'ять даних (EEPROM) організована таким чином,

що вміст кожного байта окремо може бути зчитаний або записаний. Кількість циклів перезапису енергонезалежної пам'яті перевищує 100 тисяч. Енергонезалежна пам'ять призначена для зберігання налаштувань та конфігурації програми, тобто тих даних, які повинні зберігатися під час зникнення живлення.

Зчитування та запис даних в EEPROM, як правило, здійснюється за допомогою використання відповідних регістрів з області службових регістрів SRAM. Як правило, це:

- регістр адреси при зверненні до EEPROM;
- регістр даних, зчитаних/записаних у EEPROM;
- регістр керування читанням-записом EEPROM.

При реалізації автомата з гнучкою логікою в мікроконтролері мікрокоманди часового автомата записуються в пам'ять мікрокоманд. На початку роботи завантажується адреса початкової мікрокоманди (фіксована адреса). Мікрокоманда з пам’яті завантажується в регістр мікрокоманд. Після зчитування вхідних сигналів (X) з відповідною затримкою видаються вихідні сигнали (Y) та на основі аналізу вхідних сигналів обирається адреса наступної мікрокоманди (перехід у наступний стан). Затримка в кожній мікрокоманді автомата Мура відповідає таймауту часового автомата. Обробник адрес та затримок реалізує час знаходження автомата у поточному стані та обчислює фізичну адресу наступної мікрокоманди у пам’яті мікрокоманд. За цією адресою обирається наступна мікрокоманда и записується в регістр мікрокоманд. І цикл повторюється знову (рис. 3.7).

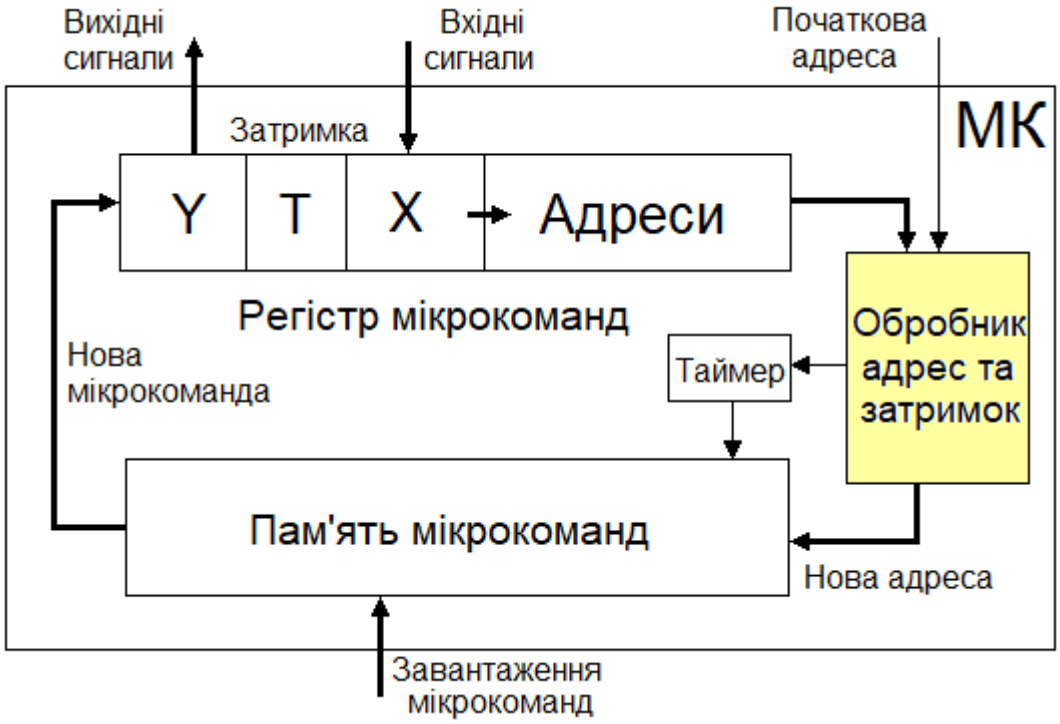


Рисунок 3.7 – Структура мікроконтролерного автомата з гнучкою логікою

3.3 Програмування мікроконтролера в форматі автоматного шаблону

При традиційному описі алгоритму функціонування цифрових пристроїв логічного управління в мікроконтролерах одним із стилів написання коду є стиль автоматного програмування. Суть автоматного програмування полягає у відділенні опису логіки поведінки (за яких умов необхідно виконати ті чи інші дії) від опису його семантики (власне сенсу кожної з дій). В автоматному програмуванні в якості базового використовується поняття «стан».

Автоматні програми строго структуровані і в них виділено три види функцій: функції переходів, функції виходів та функції реалізації затримок і переходу до нового стану. Автоматні програми строго шаблонізовані з використанням операторів багатопозиційного вибору (switch, case), умовних операторів (if, select) та функцій реалізації таймера або фронту (синхросигналу Clk). Автоматні програми інваріантні до способу кодування

(мові програмної реалізації C, JScript тощо). Код, написаний у стилі автоматного програмування, зазвичай називається «автоматний шаблон» [7].

Реалізація класичного автоматного шаблону виконана на мові C з використанням Keil v5 IDE на базі мікроконтролера STM32 F103C8T6. Дана реалізація може бути портована на будь-яке інше сімейство даних процесорів.

Початок реалізації відбувається зі створення макросів #define, які містять в собі дані щодо кожного стану автомата (a0-a4), значень затримок (timeout) у кожному стані (T0-T4) (лістинг 3.1).

Також макросами дуже зручно псевдонімізувати виклики функцій, певні вирази тощо. В даній реалізації використовуються макроси для більш стилізованого відображення виходів (y1-y3) в частині опису функції виходів (лістинг 3.2).

Лістинг 3.1 – Директиви препроцесора #define станів автомата та їх затримок

```
//States          //Timeouts in ms
//                //
#define a0  0     #define T0  1000  //Timeout for a0
#define a1  1     #define T1  3000  //Timeout for a1
#define a2  2     #define T2  1000  //Timeout for a2
#define a3  3     #define T3  5000  //Timeout for a3
#define a4  4     #define T4  15000 //Timeout for a4
```

Лістинг 3.2 – Директиви препроцесора #define псевдонімів виклику функцій

```
//Outputs
//
#define Y1_ON   HAL_GPIO_WritePin( GPIOA, GPIO_PIN_5, GPIO_PIN_SET );   //Define for y1 output OFF function on GPIO
#define Y2_ON   HAL_GPIO_WritePin( GPIOA, GPIO_PIN_6, GPIO_PIN_SET );   //Define for y2 output OFF function on GPIO
#define Y3_ON   HAL_GPIO_WritePin( GPIOA, GPIO_PIN_7, GPIO_PIN_SET );   //Define for y3 output OFF function on GPIO
#define Y1_OFF  HAL_GPIO_WritePin( GPIOA, GPIO_PIN_5, GPIO_PIN_RESET ); //Define for y1 output ON function on GPIO
#define Y2_OFF  HAL_GPIO_WritePin( GPIOA, GPIO_PIN_6, GPIO_PIN_RESET ); //Define for y2 output ON function on GPIO
#define Y3_OFF  HAL_GPIO_WritePin( GPIOA, GPIO_PIN_7, GPIO_PIN_RESET ); //Define for y3 output ON function on GPIO
```

Основна частина програми також має певний макрос (DEBUG_MODE), який дозволяє не використовувати апаратну імплементацію при переході з початкового стану у стан a1 (запуск тепловентилятора до певних обертів) та при переході зі стану a3 у стан a4 (перехід від повного нагріву до

охолодження). Такий спосіб дозволяє зекономити на кнопці та температурному сенсорі і мінімально впливає на кількість використаної пам’яті мікроконтролера.

Опис основної частини коду має статичні глобальні змінні. Статичні змінні використовуються для тих цілей, якщо змінні не мають використовуватися в інших одиницях трансляції опису (одиницею трансляції є «*.c» файл), а глобальне створення цих змінних дозволяє використовувати їх без ініціалізації (початкова ініціалізація необов’язкова, так як всі глобальні змінні автоматично ініціюються «0») по всій одиниці трансляції.

Ключове слово volatile дає розуміння компілятору, що значення змінних не має бути прихованим чи незрозумілим частинам коду, які переривають виконання основної програми. Це важливий ідентифікатор при роботі з перериваннями.

Змінними з усіма ключовими словами, які описані вище, є змінні поточного та наступного станів, змінна, яка зберігає таймаут та змінні, які використовуються при автоматизованому тестуванні без участі додаткових апаратних реалізацій (лістинг 3.3).

Лістинг 3.3 – Макрос DEBUG_MODE та статичні глобальні змінні

```
/* USER CODE BEGIN PD */
#define DEBUG_MODE //define for testing mode (without transitions with external buttons and sensors)
/* USER CODE END PD */
/* USER CODE BEGIN PV */
static volatile uint8_t state;
static volatile uint8_t nextState;

static volatile uint16_t Timeout; //volatile identifier for using the variable value in timer interrupt

#ifdef DEBUG_MODE
  static volatile uint8_t Onn; //variable for transition launching without external buttons and sensors
  static volatile uint8_t tm; //variable for transition to a3 state without external buttons and sensors
#endif

/* USER CODE END PV */
```

Перед початком виконання основної частини коду автоматного шаблону необхідно виставити межі температур, за якими можна відслідкувати чи відбулося нагрівання або охолодження. Ці значення

вносяться у константні змінні, значення яких встановлюються при ініціації. Після цього їх значення автоматично не можуть бути змінені.

Оскільки апаратна реалізація з температурним сенсором може бути необов'язкова, то необхідно програмно імплементувати нагрів та охолодження системи. Для цього з використанням директиви #ifdef створена змінна значення якої буде мати поточну температуру. Дана директива обов'язково використовується з директивою #endif (лістинг 3.4). Такий спосіб даж розуміння компілятору, що на етапі препроцесингу (конкатенація всіх директив препроцесора до файлу з розширенням «*.c» - #include, #define, #ifdef, #ifndef, тощо) має бути створений макрос, сигнатура якого відповідає сигнатурі директиви #ifdef (DEBUG_MODE). Якщо умова виконується, то і виконується даний код.

Лістинг 3.4 – Опис локальних змін контролю температури

```
/* USER CODE BEGIN 1 */
const uint8_t TEMP_TO_HEAT  = 20; //const value for cool temperature sensor
const uint8_t TEMP_TO_COOL  = 50; //const value for heat temperature sensor

#ifdef DEBUG_MODE
  uint8_t debugTemp = TEMP_TO_HEAT; //variable for getting the temperature without external sensor
#endif
/* USER CODE END 1 */
```

Надалі необхідно ініціювати всі необхідні переривання та дозволити початок відліку до переривання від таймеру, за яким відбувається контроль роботи кожного часу за певним таймаутом (лістинг 3.5). Тобто таким чином системі надається розуміння, що функція затримок має бути опрацьованою.

Лістинг 3.5 – Ініціалізація переривань та дозвіл на обробку переривання від таймеру

```
/* Initialize interrupts */
MX_NVIC_Init();                 //Allow all the necessary interrupts
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim3); //Allow to start the transition function
```

Основна програмна реалізація автоматного шаблону має три частини: функцію переходів, функцію виходів та функцію затримок. Функції переходів та виходів (ФПВ) опрацьовуються до тих пір, поки система має живлення.

Всі функції мають опис класичного SWITCH перемикання. Оскільки в даній системі не критичний час виконання (IF перемикання має незначну перевагу через меншу кількість асемблерних команд), тому використання даного варіанту більш стилізоване.

Особливістю функції переходів є додаткові тестові частини коду, які описані за допомогою директиви #ifdef. Початок роботи ініціюється вмиканням теоретичної «кнопки» в стані a0. Також такі частини коду знаходяться в описі стану a2, коли треба відслідкувати, чи нагрівся елемент чи ні, та у стані a3, коли для тестування всіх станів автомата необхідно перейти у стан a4 та скинути у початкові значення всі сигнали (Onn, tm, debugTemp) (лістинг 3.6).

Лістинг 3.6 – Функція переходів із тестовими частинами коду

```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* TRANSITIONS FUNCTION START */
  switch(state)
  {
    case a0:

      //Special part of code that initiate the transition
      //        without extrnal buttons and sensors
      #ifdef DEBUG_MODE
        Onn = 1;
      #endif

      if(Onn)
      { nextState = a1; }
      else
      { nextState = a0; }

    break;

    case a3:

      //Test code for direct transition from state a3 to state a4
      //
      #ifdef DEBUG_MODE
        Onn = 0;
        tm  = 0;
        debugTemp = TEMP_TO_HEAT;
      #endif

      if( Onn && tm )
      { nextState = a3; }
      else if ( Onn && !tm )
      { nextState = a2; }
      else
      { nextState = a4; }
    break;
```

```
    case a2:

      //Special code for debug mode
      //helps to count the necessary temp value (50 degrees by Celcius)
      //and get the correct nextState value without external buttons
      //                                                    and sensors
      #ifdef DEBUG_MODE
       if( debugTemp >= TEMP_TO_COOL )
       {
         tm = 1;
       }
       else
       {
         debugTemp += TEMP_TO_HEAT/2; //each 10 degrees
         tm = 0;
       }
      #endif

      if( Onn && tm )
      {
        nextState = a3;
      }
      else if ( Onn && !tm )
      {
        nextState = a2;
      }
```

Функція виходів має класичний опис та відображає виходи в поточному стані (лістинг 3.7).

Лістинг 3.7– Функція виходів

```
/* OUTPUTS FUNCTION START */
switch(state)
{
  case a0:
    Y1_OFF;Y2_OFF;Y3_OFF;
  break;

  case a1:
    Y1_ON;Y2_OFF;Y3_OFF;
  break;

  case a2:
    Y1_ON;Y2_ON;Y3_ON;
  break;

  case a3:
    Y1_ON;Y2_OFF;Y3_ON;
  break;

  case a4:
    Y1_ON;Y2_OFF;Y3_OFF;
  break;

  default:
    Y1_OFF;Y2_OFF;Y3_OFF;
  break;
}
/* OUTPUTS FUNCTION END */
```

В обробнику переривань також існує тестова частина коду з відображенням сигналів Onn та tm на портах введення/виведення мікроконтролера, які можна буде відображатися на вейвформі (лістинг 3.8).

Лістинг 3.8 – Функція виведення сигналів Onn та tm

```
#ifdef DEBUG_MODE
    HAL_GPIO_WritePin( GPIOA, GPIO_PIN_3,    (GPIO_PinState)Onn );
    HAL_GPIO_WritePin( GPIOA, GPIO_PIN_4,    (GPIO_PinState)tm  );
#endif
```

Для того, щоб відлік до наступного переривання мав відповідати значенню таймаута, необхідно занести це значення в регістр таймеру .

Також функція переходів-виходів перериваються після певного проміжку часу. В цей час відбувається переривання і управління передається обробнику переривань, в якому знаходиться функція затримок та ініціація переходу у наступний стан (лістинг 3.9).

Лістинг 3.9 – Функція затримок встановлення нового таймауту таймера

```
/* DELAYS FUNCTION START */
state = nextState; //transit
           to nextState value
//select the necessary
// Timeout for current state
switch(state)
{
  case a0:
    Timeout = T0;
  break;

  case a1:
    Timeout = T1;
  break;

  case a2:
    Timeout = T2;
  break;

  case a3:
    Timeout = T3;
  break;

  case a4:
    Timeout = T4;
  break;

  default:
    Timeout = T0;
  break;
}

/* DELAYS FUNCTION END */
```

Повний програмний код автоматного шаблону наведений у Додатку В.

3.4 Програмування мікроконтролера для реалізації автомата з гнучкою логікою

Розглянемо особливості реалізації автомата з гнучкою логікою. Реалізація, що має опис нижче отримала багато назв – автомат з гнучкою логікою, парсер команд, двигун. Дана реалізація також спроєктована на мові С з використанням Keil v5.32 IDE на базі мікроконтролера STM32 F103C8T6. Як і попередньо розроблена реалізація, вона має всі можливості портуватися до будь-якого іншого сімейства мікроконтролерів фірми STMicroelectronics.

У файлі main.h необхідно задекларувати невелику кількість макросів для подальшого розуміння кількості станів, чи є безумовний перехід, перетворювач затримок із секунд в мілісекунди для перезавантаження таймеру переривань.

Макроси OUTPUT_MISSING_NUMBER та INPUT_MISSING_NUMBER необхідні для даної реалізації для заповнення уніфікованої структури даних такими числами, які парсером команд не будуть опрацьовані.

Лістинг 3.10 – Директиви препроцесора #define для реалізації

```
/* USER CODE BEGIN Private defines */

#define STATES_COUNT               5
#define DEFAULT_TRANSITION         0
#define ONE_SEC                 1000 //Timeout to seconds definition

#define OUTPUT_MISSING_NUMBER      3 //Special define for output that might not be shown on GPIO
#define INPUT_MISSING_NUMBER      15 //Special define for input statements that might not be cheked on states

/* USER CODE END Private defines */
```

Основний файл виконання main.c містить небагато коду. Основні його частини – проектування пам’яті (memory mapping), створення структури даних за методом проектування, декларування необхідних змінних, опис парсеру команд та переривання від таймеру для реалізації переходу в наступний стан та завантаження таймауту цього стану у таймер до наступного переривання.

Перша помітна зміна реалізації – це проектування пам’яті. Її ідея зображена на рис.2.4 та 3.6. Дане проектування реалізоване за допомогою універсальної структури – бітових полів (bit fields). Її реалізація дуже схожа з результатом, зображеним на рис.3.6, але має певні відмінності. По-перше, всі команди мають однакову довжину. По-друге, додаткова кількість полів не завжди збільшує розмір даної пам’яті. Оскільки компілятор самостійно визначає розмір структури за принципом байтового виділення пам’яті, то, наприклад, використання трьох полів розміром у 3 біта все одно будуть зарезервовані в пам’яті як 2 байти (16 біт). Тому біти, що не використовуються можуть бути використані для інших полів.

Бітові поля типу curStateMicroCommand[] задекларовані спеціальним ключовим словом typedef. Це дає розуміння компілятору, що весь код, який написаний до типу даних (curStateMicroCommand[STATES_COUNT]) буде псевдонімізований як цей тип (лістинг 3.11). Як результат всі наші команди будуть мати вигляд бітових полів, розмір кожної з яких сягає 7 байт (56 біт).

Результатом проектування є структура даних – масив бітових полів на п’ять значень (скільки станів – стільки і мікрокоманд).

Лістинг 3.11 – Memory mapping

```
/*
 * bitfield struct as the COMMAND PARSER
 * each field has the command bitcode sequence
*/
typedef struct
{
  uint8_t curState          :3;
  uint8_t curStateDelay     :4;
  uint8_t outputsAmount     :2;
  uint8_t arcAmount         :3;
  uint8_t firstOutputValue  :2;
  uint8_t firstOutputDelay  :2;
  uint8_t secOutputValue    :2;
  uint8_t secOutputDelay    :2;
  uint8_t thirdOutputValue  :2;
  uint8_t thirdOutputDelay  :2;
  uint8_t firstInputValue   :4;
  uint8_t firstNextState    :3;
  uint8_t secInputValue     :4;
  uint8_t secNextState      :3;
  uint8_t thirdInputValue   :4;
  uint8_t thirdNextState    :3;
}curStateMicroCommand[STATES_COUNT];
```

На лістингу 3.12 зображений макрос для можливості тестування без використання кнопки запуску та температурного сенсору та макроси станів. Макроси станів використовуються у тест-режимі для зміни вхідних сигналів, які ініціюють перехід у наступний стан.

Лістинг 3.12 – Макрос DEBUG_MODE та макроси станів

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DEBUG_MODE //Special define for unit test execution without sensors

#ifdef DEBUG_MODE //Special state defines for unit test executions without sensors
  #define a0 0
  #define a1 1
  #define a2 2
  #define a3 3
  #define a4 4
#endif
/* USER CODE END PD */
```

Наступною особливістю реалізації автомата з гнучкою логікою – це структура даних, що побудована за проектуванням пам’яті (рис.х.4).

Варто зазначити, що дана структура даних введена мануально для представлення роботи тільки двигуна. Всі дані вводяться в десятинній формі, за умови відповідності кількості біт, які займають дані числа. Як зазначено вище особливістю даної структури даних є те, що не зважаючи на рис.3.6 кожна команда має однакову кількість бітів, але ті біти, які не мають бути оброблені отримують максимально можливе значення (так як всі бітові поля, за правилом, мають мати тільки позитивні значення) і за допомогою макросів OUTPUT_MISSING_NUMBER та INPUT_MISSING_NUMBER надалі опускаються двигуном.

Лістинг 3.13 – Структура даних з описом мікрокоманд

```
/* USER CODE BEGIN PV */

]/*
  * Memory mapping example
  * Each bitfield (a structure with fields that have concrete bit amount)
  * decribes the microcommand section that will be processed by the command parser
  * (microcommand execution engine)
  * The memory mapping is unified structure with possibility not to operate with
  * microcommand sections that have the maximum field value
  * e.g. if the firstOutputDelay (2 bits) field has the value of 3
  * the engine will not set the output value to "1" on GPIO
-*/
 static volatile curStateMicroCommand STATES =
]{
   { 0, 1,0,2, 3,3,3,3,3,3, 8,0,4,1,15,7  },
   { 1, 3,1,2, 1,0,3,3,3,3, 4,2,8,0,15,7  },
   { 2, 1,3,3, 1,0,2,0,3,0, 6,2,5,3,8,4   },
   { 3, 5,2,3, 1,0,3,3,3,0, 6,2,5,3,8,4   },
   { 4,15,1,1, 1,0,3,3,3,3, 0,0,15,7,15,7 }
 };
```

Опис необхідних статичних глобальних змінних лишається (поточний стан, наступний стан та таймаут стану). Додатковою змінною є OnnAndTm, яка використовується для тестування роботи системи без натискань додаткової кнопки та отримання даних із сенсора температури (лістинг.3.14). Дана змінна може зберігати таке ж саме значення, як частина команди –

бітове поле InputValue при порівнянні з якою і відбувається збереження значення наступного стану.

Лістинг 3.14 – Статичні глобальні змінні

```
static volatile uint8_t state;      //current state
static volatile uint8_t nextState;  //next state
static volatile uint16_t Timeout;   //timeout in seconds

#ifdef DEBUG_MODE
  static volatile uint8_t OnnAndTm; //Special variable of input signals for unit test executions without external sensors
#endif
```

Обробник команд (парсер команд, двигун, автомат із гнучкою логікою) розпочинає свою роботу із тестової частини коду (лістинг 3.15). Кожна ітерація циклу дає розуміння системі, які вхідні дані мають бути збережені для подальшого переходу у наступний стан та імітує режим нагрівання у стані a2. Дані зберігаються у змінну OnnAndTm та debugTemp, яка в свою чергу також перевіряється у цьому тестовому коді.

Лістинг 3.15 – Тестова частина коду

```
/* CODE FOR UNIT TEST WITHOUT SENSORS */
#ifdef DEBUG_MODE
  if( a0 == STATES[state].curState )
  {
    //Initiate the transition to the next state
    // Onn = 01; tm = 00 so the OnnAndTm will be 0100 (4)
    //
    OnnAndTm = 4;
  }

  if( a2 == STATES[state].curState )
  {
    //Unit test to check the heating and initiate the transition from a2 to a3
    //
    if( debugTemp >= TEMP_TO_COOL )
    {
      OnnAndTm = 5;
    }
    else
    {
      OnnAndTm   = 6;
      debugTemp += TEMP_TO_HEAT/2;
    }
  }

  if( a3 == STATES[state].curState )
  {
    //Inititate the transition from a3 to a4 with set the variables to the default value
    //OnnAndTm = 1000 (nextState should be a0); debugtemp = 20;
    //
    OnnAndTm  = 8;
    debugTemp = TEMP_TO_HEAT;
  }
#endif
/* END CODE FOR UNIT TEST WITHOUT SENSORS*/
```

Перша основна частина обробника команд – це перевірка вхідних значень. Кожне вхідне значення поточного стану у команді STATES[state]. {first,sec,third}InputValue порівнюється з макросом відсутності оперування (INPUT_MISSING_NUMBER) та макросом переходу за замовчуванням або безумовним переходом (DEFAULT_TRANSITION). Якщо жодна з перевірок не спрацьовує, то результатом є автоматичний перехід у стан a4, з якого відбувається перехід у стан a0 за замовчуванням. Логічно допустити, що

щось сталося із живленням. Тобто відбувається вимкнення тепловентилятора до повної зупинки і після певного таймаута цикл повторюється або система знаходиться у початковому стані до моменту включення системи – OnnAndTm (лістинг 3.16).

Лістинг 3.16 – Функція переходів

```
/* INPUT STATEMENTS WITH TRANSITION */

//Checking whether the OnnAndTm should be processed to get the nextStateValue
//All the InputValue fields should be processed!
//
if( INPUT_MISSING_NUMBER != STATES[state].firstInputValue || DEFAULT_TRANSITION == STATES[state].firstInputValue )
{
  nextState = STATES[state].firstNextState;
}

else if( INPUT_MISSING_NUMBER != STATES[state].secInputValue || DEFAULT_TRANSITION == STATES[state].secInputValue )
{
  nextState = STATES[state].secNextState;
}

else if( INPUT_MISSING_NUMBER != STATES[state].thirdInputValue || DEFAULT_TRANSITION == STATES[state].thirdInputValue)
{
  nextState = STATES[state].thirdNextState;
}

else //default state
{
  nextState = a4;
}
/* END INPUT STATEMENTS WITH TRANSITION */
```

Друга основна частина коду – це обробка вихідних значень, які мають бути відображені у поточному стані. Спочатку відбувається порівняння значень затримок STATES[state].{first,sec,third}OutputDelay з макросом OUTPUT_MISSING_NUMBER. Якщо умова виконується, то це означає, що вихідний сигнал, який збережено у структурі даних (STATES[state].{first,sec,third}OutputValue) має бути відображено на порті виведення мікроконтролера (GPIO). Інакше значення сигналу на порті має бути «0» (лістинг.3.17).

Обробник переривань таймера описує всі ті ж функції, що й у класичному автоматному шаблоні (перехід із стану в стан, визначення затримки стану та перезавантаження таймера, виведення вхідних значень в якості тесту). Проте визначення затримки за допомогою проєктування пам’яті зводиться до одного рядка коду, який заносить значення бітового поля поточного стану (STATES[state].curStateDelay) у змінну Timeout.

Лістинг 3.17 – Функція виходів

```
/* GPIO OUTPUT */

//Checking and write the output value to the GPIO
//All the OutputDelay fields should be processed!
//
if( OUTPUT_MISSING_NUMBER != STATES[state].firstOutputDelay )
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
}
else
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
}

if( OUTPUT_MISSING_NUMBER != STATES[state].secOutputDelay )
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_SET);
}
else
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_RESET);
}

if( OUTPUT_MISSING_NUMBER != STATES[state].thirdOutputDelay )
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);
}
else
{
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET);
}
/* END GPIO OUTPUT */
```

Оскільки в цьому полі значення таймаута зберігається у секундах, то необхідно його перевести у мілісекунди для перезавантаження лічильника таймера до наступного переривання. Відбувається це за допомогою множення поля на макрос ONE_SEC (1000). Лічильник таймера перезавантажується, дозвіл на обробку переривання оновлюється і система далі продовжує свою роботу. Ще однією особливістю цього коду є отримання значення сигналів Onn та tm з однієї змінної (OnnAndTm). Оскільки за рис.3.4 встановлено, що позитивне значення сигналу Onn та tm є значення двох біт «01», то як висновок маємо зазначити, що третій біт змінної OnnAndTm сигналізує значення сигналу Onn, а перший біт змінної – сигналу tm. Отримання значень відбувається за допомогою операцій логічного І та бітового зсуву вправо (лістинг 3.18).

Лістинг3.18 – Функція переходу в наступний стан та визначення затримки стану

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) //Timer interrupt handler(callback)
{
  if(htim->Instance == TIM3)
  {
    HAL_TIM_Base_Stop_IT(&htim3); //Stop the timer and forbid the interrupt (for insurance)

    state = nextState; //Initiate the transition
    Timeout = STATES[state].curStateDelay * ONE_SEC; //Get the state timeout and convert it to the ms

    /*
     * Special part of code to show on GPIO the OnnAndTm values
     * The third bit of variable shows the Onn signal result (0 or 1)
     * The first bit of variable shows the tm signal result (0 or 1)
    */
    #ifdef DEBUG_MODE
      //Write the Onn value with parsing the third bit and tm value with parsing the first bit
      //
      HAL_GPIO_WritePin( GPIOA, GPIO_PIN_3, (GPIO_PinState)((OnnAndTm&4)>>2) );
      HAL_GPIO_WritePin( GPIOA, GPIO_PIN_4, (GPIO_PinState)(OnnAndTm&1) );
    #endif

    __HAL_TIM_SET_AUTORELOAD(&htim3, Timeout); //Reload the timer for interrupt restarting

    HAL_TIM_Base_Start_IT(&htim3); //return the interrupt mode
  }
}
/* USER CODE END 4 */
```

3.5 Відображення інформації та аналіз результатів

При отриманні налагоджувальної інформації одним з основних критеріїв є її вигляд. Це можуть бути як звичайні світлодіоди, так і промислові комп’ютери, які використовуються як тестувальні стенди. Але оскільки логіка роботи цифрових схем базується на передачі цифрових та аналогових сигналів (у вигляді «0» та «1» та змін напруги хвилями за певний проміжок часу), є необхідність отримання цієї інформації у такому ж вигляді, то існує певний вид відображення як часова діаграма (waveform). Дуже популярними пристроями, які дозволяють отримати такий вигляд інформації є логічні аналізатори [9].

Логічні аналізатори – це цифро-аналогові пристрої, які використовуються для тестування складних цифрово-аналогових схем. Вони призначені для оцінки та відображення як цифрових, так і аналогових сигналів (в деяких моделях).

Інженери використовують їх для проектування, оптимізації та налагодження обладнання, яке використовується в прототипах цифрових систем. Такий спосіб відлагодження дозволяє усунути проблеми з несправними системами. Основне завдання логічного аналізатора - фіксувати та відображати послідовність цифрових подій (сигналів за певний проміжок часу роботи системи). Після захвату дані відображаються у вигляді часових діаграм для відображення даних, що передаються (декодований трафік, передача даних, флаги дозволу або заборони, робота арбітражу інформації (мультиплексор)). Деякі логічні аналізатори можуть захопити новий набір даних і порівняти його з отриманим раніше.

В даний час на ринку представлені в основному три типи логічних аналізаторів.

1. Модульні логічні аналізатори. Ці логічні аналізатори постачаються з базовим блоком, мейнфреймом і модулем логічного аналізатора. Базовий блок (ще його називають шасі) містить елементи управління, комп'ютер управління, дисплей та кілька слотів. Ці слоти використовуються для розміщення фактичного програмного забезпечення для збору даних.

2. Портативні логічні аналізатори Портативні логічні аналізатори часто називають автономними логічними аналізаторами. В цьому виді аналізаторів кожен компонент інтегрований в єдиний пакет. Незважаючи на меншу продуктивність, їх більш ніж достатньо для загальних цілей.

3. Логічні аналізатори на базі персональних комп’ютерів (ПК). Ці логічні аналізатори працюють при підключенні до ПК через з'єднання USB або Ethernet. Сигнали, що було захвачено, передаються в програмне забезпечення на комп'ютері.

Логічні аналізатори та осцилографи схожі між собою, але все одно мають різні параметри, які описані нижче.

1. Логічний аналізатор записує у пам’ять всі дані перед їх відображенням за встановлений проміжок часу роботи системи. Осцилограф багаторазово зберігає і показує невеликі знімки. Такі знімки та відображення

називаються семплами.

2. Логічні аналізатори мають функцію, яка дозволяє користувачам переміщатися по потенційно довгим записам. В осцилографі сигнали (дані для відображення) представлені у реальному часі.

3. Логічний аналізатор виконує вимірювання від точки до точки збору даних. Осцилограф вимірює амплітуду, частоту та синхронизацію сигналу (період).

4. Логічні аналізатори мають багато функцій, унікальних для цифрових систем. Прикладом цього є аналізатори протоколів даних (UART, SPI, I2C, 1-Wire). Осцилографи мають деякі алгоритми перетворення даних у реальному часі, такі як швидке перетворення Фур'є (ШПФ).

5. Логічні аналізатори мають складні системи програмного та апаратного запуску, які використовуються для збору та фільтрації даних. Осцилографи мають прості порогові або широтно-імпульсні тригери, які використовуються для зображення стійкої форми сигналу.

На рис. 3.8 зображена структурна схема роботи логічного аналізатора.

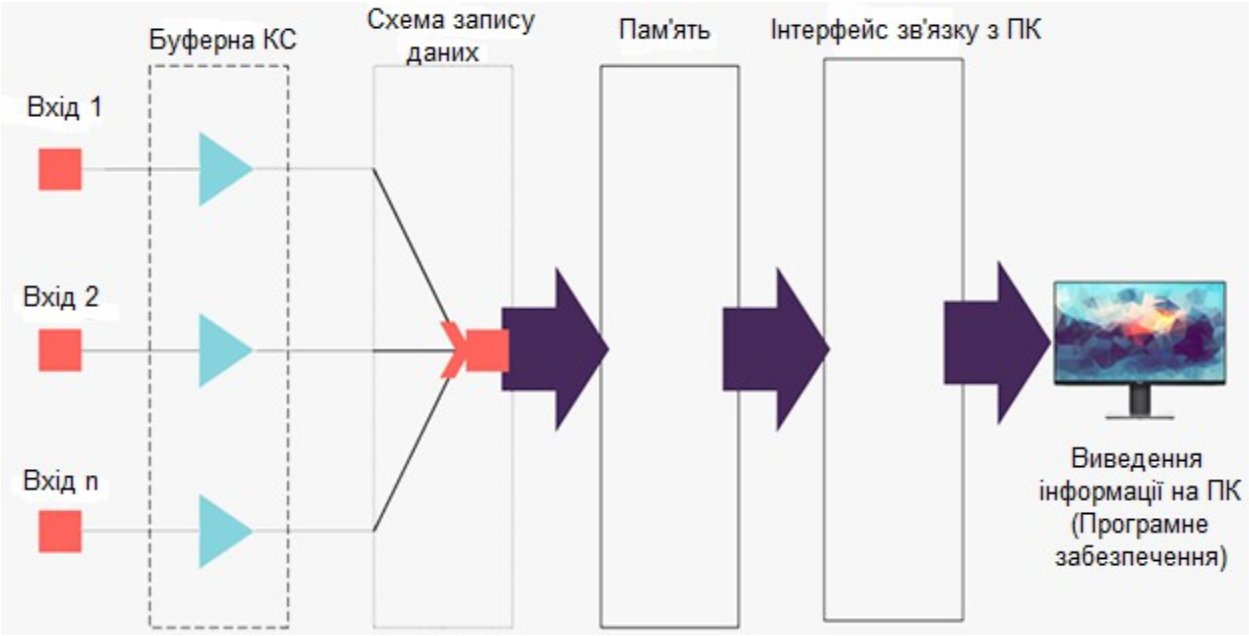


Рисунок 3.8 – Структурна схема логічного аналізатора

Напруга з кожного входу через буферну комбінаційну схему (КС) з певним показником напруги (проходження через буфер може незначно змінити напругу у меншу сторону або стабілізувати її) потрапляє у схему запису даних (пристрій запису), де фіксується логічний стан кожного входу.

Після вимірів логічних станів на всіх входах (навіть не підключених до пристрою, які майже завжди перебувають у стані «1»), дані зі схеми буферизуються у пам'ять (записуються блочним типом). Перед записом даних у пам’ять пристрій запису виконує компресію (стискання) для оптимізації обмеженого розміру пам'яті. Варто зазначити, що не всі логічні аналізатори стискають дані, тому процес збирання даних різних видів аналізаторів може бути дуже швидким або, навпаки, повільним.

Далі потрапляють на інтерфейс зв'язку з ПК, на якому запущено програмне забезпечення для відображення записаних даних. Уже в цьому програмному забезпеченні виконується декодування протоколів, пошук шумів, нестабільності передачі даних.

Важливу роль у використанні логічних аналізаторів виконує програмне забезпечення (ПЗ), адже у ньому відбувається декодування та аналіз отриманих даних. Винятком можуть бути автономні портативні пристрої, які відображають записані послідовності безпосередньо на екран. Програмне забезпечення доступне на сайтах виробників логічних аналізаторів, яке можна встановити та запустити, не маючи на руках потрібного приладу. Використовуючи тестові режими або режими симуляції, можна побачити, як виглядатимуть сигнали, отримані з аналізатора.

Для отримання даних з системи, що розробляється, буде використано логічний аналізатор Saleae Logic 8 (рис. 3.9). У даного аналізатора логіки більше ніж достатньо для отримання сигналів з каналів або передачі даних протоколів. А його ціна у 10 USD дозволяє визнати його найдешевшим способом аналізу трафіку та даних у будь-яких проектах з невеликою частотою. Даний логічний аналізатор підтримується ПЗ, що безкоштовно надається компанією Saleae.

Рисунок 3.9 – Логічний аналізатор Saleae Logic 8

Нещодавно ПЗ Saleae (Logic) мало оновлення до другого покоління (версія 2.3.40). В оновленні було повністю змінено інтерфейс, який став ще простішим та звівся до суто до маніпуляцій з меню (рис. 3.10).

Інтерфейс складається з відображення назв каналів, які можна перейменувати, результатів вибірки за певний проміжок часу, який можна аналізувати збільшуючи чи зменшуючи колесом миші. Та невеликого меню, за яким і відбувається налаштування отримання даних користувачем.Також при наведені миші на певну частину вибірки можна побачити проміжок часу та частоту між зміною фронтів (рис. 3.10).

Основне меню, зображене на рис.х.8 дозволяє обрати кількість каналів для аналізу даних (не тільки дискретних, так як деякі логічні аналізатори мають змогу відслідковувати дані з АЦП), обрати режим захвату даних (Looping, Timer, Trigger) та кількість пам’яті, яку може бути виділено для семплів. Пам’ять є єдиним критерієм автоматичного запису даних у режимі Looping. В режимі Timer можна записувати дані як за певний проміжок часу, так і за кількості виділеної пам’яті. Режим trigger дозволяє користувачу завершити запис до появи першого фронту або імпульсу на каналі. Також час запису після захвату необхідних даних можна встановити мануально.

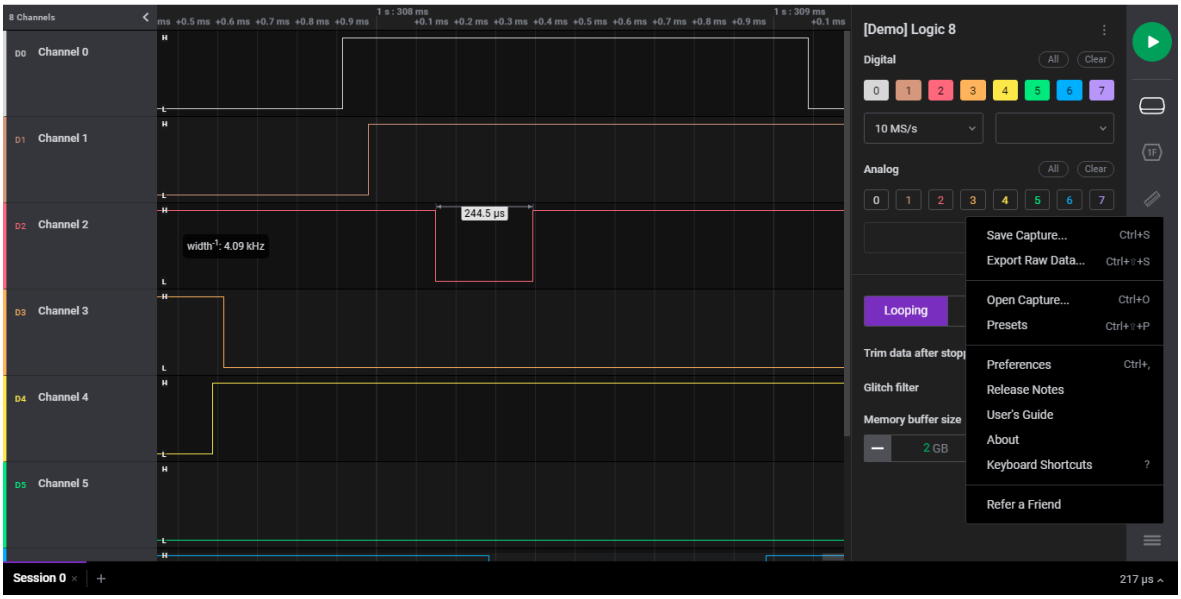


Рисунок 3.10 – Демонстрація меню ПЗ Saleae Logic

Для демонстрації працездатності мікроконтролерної моделі часового керуючого автомату Мура в форматі автоматного шаблону виконаємо обхід всіх вершин графової моделі з урахуванням часових параметрів за циклом :

a0 – a1(y1, 3 сек.) – a2(y1, y2, y3, 1 сек.) – a3(y1, y3, 5 сек.) – a2(y1, 3 сек.) – – a4(y1, 15 сек.) – a0 .

Початок роботи системи ініціюється включенням сигналу Onn (значенням логічної «1») з відображенням високого рівня сигналу на виході y1. Цей момент означає перехід зі стану a0 в стан a1.

Момент появи високого рівня сигналу на виході y2 та y3 означає перехід зі стану a1 в a2.

Поява високого рівня на сигналі tm означає нагрівання системи до встановленого максимуму температури та переходу зі стану a2 в a3.

З плином конкретного проміжку часу (таймауту T3), тестова частина коду спрацьовує та переводить сигнал Onn та tm у низький рівень, що ініціює перехід зі стану a3 в стан a4.

Перехід зі стану a4 в стан a0 відбувається падінням рівня сигналу на виході Y1. Цикл повторюється через 1 секунду затримки T0 у стані a0.

На рис. 3.11 представлений графічний результат роботи системи у

вигляді вейвформу. Виділено 5 цифрових каналів, які відображають момент включення та виключення сигналів Onn, tm, y1, y2 та y3 у відповідності до функції переходів, виходів та затримок. Детальний опис переходів та відображення часу переходів, сигналів представлений нижче. Для більш зручного відображення був взятий 2 та 3 цикл роботи системи.

Результати, відображені на waveform, повністю співпадають зі специфікацією (графова модель на рис. 3.2).

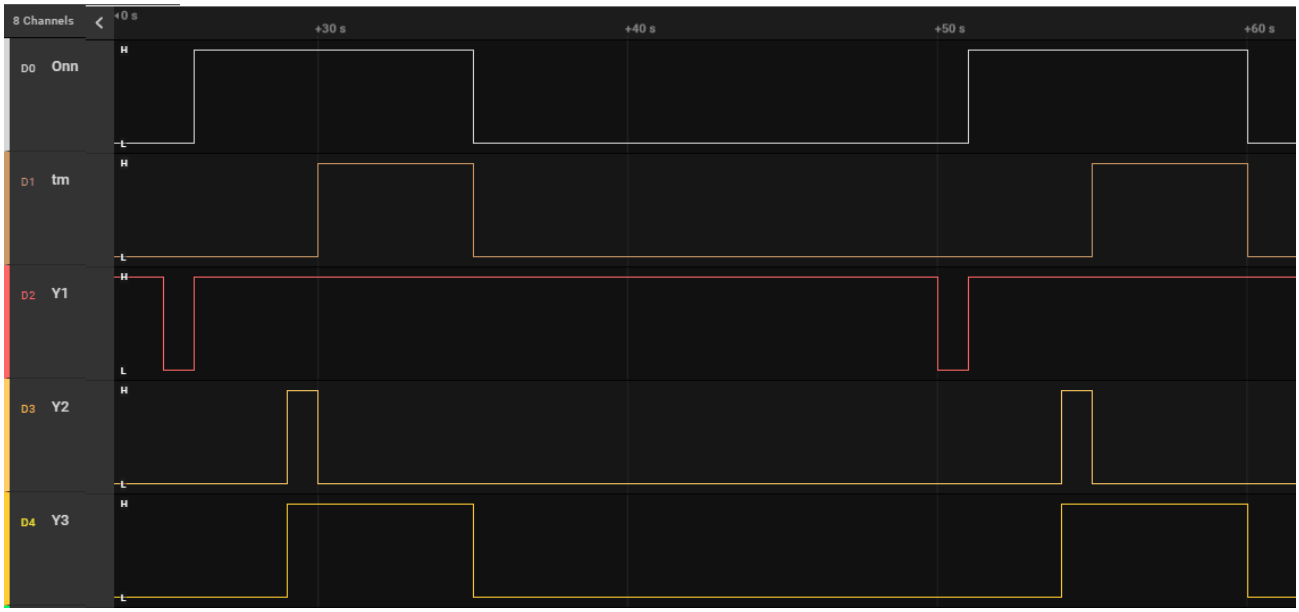


Рисунок 3.11 – Результат роботи системи у вигляді waveform

Для детальної статистики всіх переходів та затримок використано меню програмного забезпечення від компанії Saleae Measurements. Детальна демонстрація кожного стану, вихідних значень та часу переключення зображена на рис. 3.12. Воно дозволяє графічно розділити вейвформ на частини за часом та кольором, додати коментарі до кожної з виділених частин та отримати різницю часу між виділеними частинами. Недоліком є похибка у вимірюванні. Вплив на похибку є як зі сторони апаратної так і зі сторони програмної реалізації (з’єднання логічного аналізатора з мікроконтролерною системою та різні моменти переключення, системні переривання, переривання таймеру. Все це називається контекст

переключення – switching context). При детальній роботі із вейвформом (розтягування на 10-ти мікросекундний діапазон, точна вставка маркеру у момент переключення) похибка становиться мінімальною.

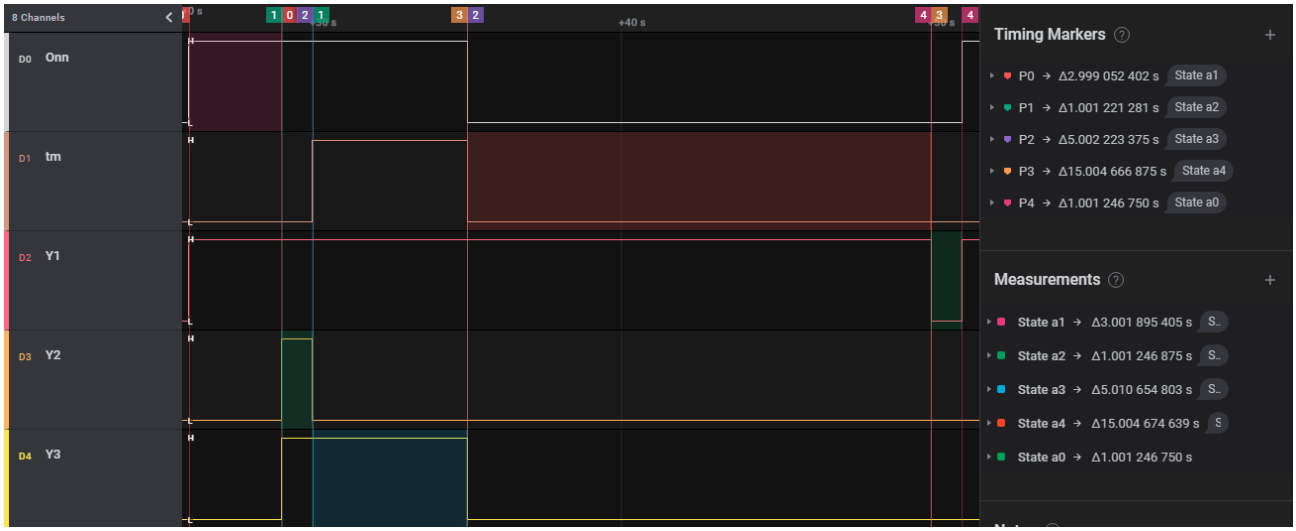


Рисунок 3.12 – Waveform роботи системи з використанням режиму Timestamps and measurements

Результатом роботи програмної моделі автомата з гнучкою логікою є вейвформ, зображений на рис.3.13. Даний результат нічим не відрізняється від вейвформи роботи автоматного шаблону, що означає повне коректне виконання всіх інструкцій в обох способах реалізації.

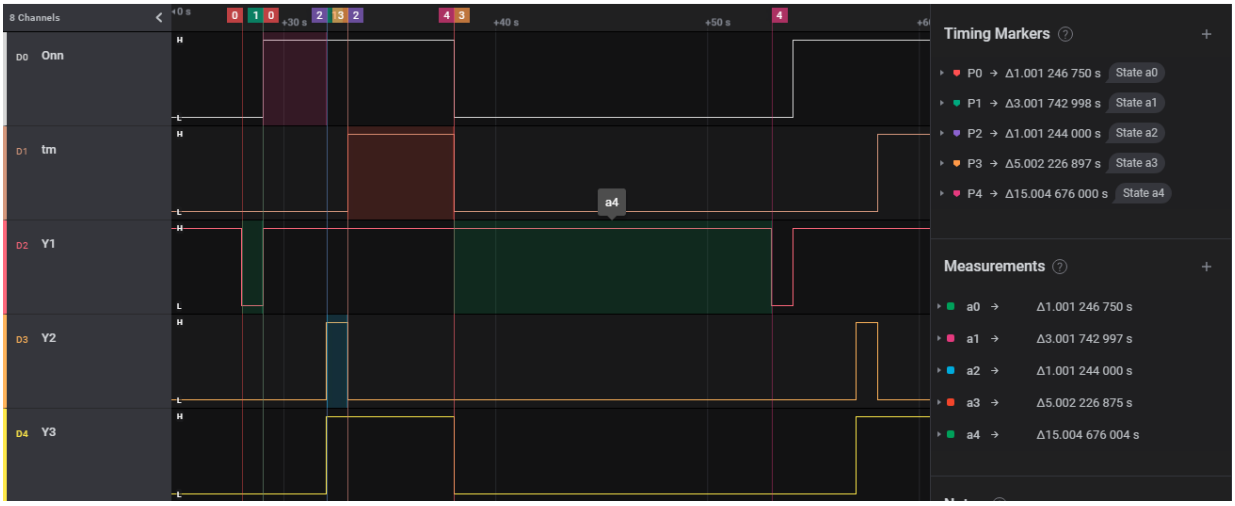


Рисунок 3.13 – Waveform роботи системи для автомату з гнучкою логікою

Оскільки обидва методи реалізації системи підтвердили свою працездатність за всіма правилами та коректність виконання всіх інструкцій, то необхідно провести порівняльну характеристику, визначити перевагами та недоліками.

Критерії оцінювання мають бути наступні.

1. Кількість зайнятої пам’яті на мікроконтролері (*.hex файл – виконавчий файл з двійковим кодом, «прошивка» для мікроконтролера).

2. Кількість строк реалізованого коду, включаючи обробник переривань від таймера, створення та виклик функцій ініціації переривань.

Перший критерій дуже легко проаналізувати завдяки реалізації даних способів у середовищі програмування Keil v5.32 IDE, який має вбудований компілятор та за потребою автоматично генерує файл прошивки.

Варто зазначити, що в даному випадку система Windows 10 For Educational, округлює розмір файлу до найбільшого цілого значення. Але найважливіший критерій – це кількість строк у файлі прошивки (табл. 3.1).

Перший критерій не має 100% ефективності, так як результатом всіх етапів компіляції є виконавчий файл, який має всі залежності файлів-заголовків (*.h), які на етапі препроцесінгу конкатенуються з усіма *.c файлами, статичними та динамічними бібліотеками (*.lib, *.dll), автоматичну конфігурацію усієї пам’яті мікроконтролера (startup file). Тому оцінити різницю між способами можливо тільки за кількістю рядків у даному файлі (рис. 3.14).

Рисунок 3.14 – Приклад команд *.hex файлу

Другий критерій можна визначити двома способами: мануальним рахунком рядків коду або за допомогою спеціальних застосунків (Sublime, Notepad++, вбудовані можливості IDE). Оскільки не всі IDE мають функціонал підрахунку власних рядків коду, а текстовий редактор обов'язково вбудований у середовище програмування, то використавши перший спосіб рахунку кількості рядків власного коду отримаємо такі значення (табл.3.1).

Таблиця 3.1 – Порівняння способів програмної реалізації за виконавчим файлом прошивки (*.hex) та довжиною коду

Спосіб реалізації	Кількість рядків коду	Об'єм файлу прошивки	Кількість рядків файлу прошивки
Класичний автоматний шаблон	133	18 кБайт (17.9)	412
Автомат із гнучкою логікою	71	18 кБайт (17.4)	400

Розглянемо переваги та недоліки програмної реалізації класичного автоматного шаблону.

Переваги:

– не потребує проектування пам'яті;

– реалізація всіх функцій з використанням єдиної SWITCH-технології;

Недоліки:

– знання та навички програмування (як реалізувати автоматний шаблон на базі мікроконтролеру);

– збільшення кількості макросів та рядків коду реалізації ФПВ пропорційно кількості станів;

– збільшення необхідної пам'яті на мікроконролері у зв'язку зі збільшенням кількості коду;

– збільшення опису функції затримок у обробнику переривань

порушує саму концепцію використання обробника переривань, необхідно повністю оптимізувати код або провести рефакторінг.

Розглянемо недоліки та переваги програмної реалізації автомату із гнучкою логікою.

Переваги:

– мала кількість макросів та опису коду (при цьому двигун змінюється тільки за кількості вхідних та вихідних значень для підтримки універсальності структури даних та єдиного стандарту опису програмної реалізації, а також реалізація опису значення затримки стану зводиться до одного рядка коду;

– необхідна інформація тільки про ТПВ та загальний вигляд мікрокоманд;

– збільшення структури даних відбувається за кількістю та обсягом мікрокоманд фіксовано (побайтно), що можна відслідкувати .

Недоліки:

– потребує проектування пам’яті та створення структури даних. Здебільшого дана структура даних має бути універсальною та може займати більше пам’яті;

Як результат універсальність програмної реалізації автоматного шаблону може мати недолік у додаткових витратах пам’яті при збільшені довжини мікрокоманди та їх кількості, але це єдиний блок коду, який постійно може змінюватися. Реалізація двигуна практично залишається статичною і залежить суто від структури даних. При цьому кількість рядків у .hex файлі не змінюється.

У програмній реалізації автоматного шаблону недоліки тісно пов’язані з кількістю станів, їх реалізації у функціях переходів, виходах та затримок. Наприклад, автомат на 20 та більше станів з різними і навіть повторюваними затримками у деяких станах вже має бути переглянутий, так як велика кількість коду у функції переходів, навіть, при заміні SWITCH-технології на IF порушують правила використання обробника переривань.

ВИСНОВКИ

Метою кваліфікаційної роботи було побудова моделі часового керуючого автомата з гнучкою логікою та реалізація цієї моделі в мікроконтролерних пристроях.

В ході виконання роботи проведений аналіз методів проектування систем логічного управління та аналіз методів побудови керуючих мікропрограмних автоматів з гнучкою логікою.

Розроблений формат мікрокоманди часового керуючого автомату Мура з гнучкою логікою та структура реалізації керуючого автомату Мура з гнучкою логікою в мікроконтролері.

Моделі часових автоматів представлені на мові програмування CI для мікроконтролера STM32F103 у формі автоматного шаблону та у формі набору мікрокоманд з примусовою адресацією для гнучкого автомата. Виконано поведінкове моделювання запропонованих моделей, результати представлені у формі часових діаграм за допомогою програмного логічного аналізатора.

Проведений порівняльний аналіз розробленої моделі мікроконтролерного автомата з гнучкою логікою з традиційною моделлю часового автомата у формі автоматного шаблону на мові програмування CI.

Наукова новизна роботи полягає у розробці формату мікрокоманд для часового керуючого автомата з гнучкою логікою, а також структура мікроконтролерного автомата з гнучкою логікою, яка реалізує зазначений формат.

Практична цінність роботи полягає в розробці мікроконтролерної програми реалізації керуючого автомата Мура з гнучкою логікою, що дозволяє не змінювати прошивку мікроконтролера при зміні алгоритму роботи керуючого автомата, а змінювати тільки дані мікропрограми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Баркалов А.А. Исследование автомата с программируемой логикой в составе цифровой информационно-управляющей системы на FPGA / А.А. Баркалов, Л.А. Титаренко, И.Я. Зеленева, С.С. Грушко – Problemele Energeticii Regionale – № 1-2 (41) – 2019 – С. 37-45.

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

3. Жмакин А.П. Архитектура ЭВМ / А.П. Жмакин. – СПб.: БХВ-Петербург, 2006. – 320 с.

4. Баранов С.И. Синтез микропрограммных автоматов (граф схемы и автоматы).– 2-е изд., перераб. и доп./ С.И.Баранов. – Л.:Энергия, 1979. – 232 с.

5. Автоматы с программируемой логикой [Электронний ресурс] / Образовательная социальная сеть KazEdu.kz. – Режим доступу: www / URL: https://www.kazedu.kz/referat/201269 – 17.09.2021 р. – Загол. з екрану.

6. Shkil A.S. Design timed FSM with VHDL Moore pattern / M.A. Miroschnyk, A.S. Shkil, E.N. Kulak, D.Y. Rakhlis, A.M. Miroshnyk, N.V. Malahov // Radio Electronics, Computer Science, Control. – 2020. – №2(53). – P. 137-148.

7. Шкіль А.С., Автоматизированное проектирование систем логического управления с использованием шаблонов автоматного программирования / А.С. Шкиль, Э.Н. Кулак, И.В. Филиппенко, Д.Е. Кучеренко, М.В. Гога. // Радіоелектроніка та інформатика – 2018. – №3. – С. 75-81.

8. Торгаев С.Н. Практическое руководство по программированию STM-микроконтроллеров / С.Н. Торгаев, М.В. Тригуб , И.С. Мусоров , Д.С. Чертихина. – Томск: Изд-во ТПУ, 2015. – 111 с.

9. Как выбрать логический анализатор [Электронный ресурс] / Суперайс. – Режим доступу: www / URL: https://supereyes.ru/articles/other/logicheskiy-analizator-kak-vybrat/ – 06.12.2021 р. – Загол. з екрану.