Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

_____*ГЮІК. 46741Х.003 ПЗ*_____
(позначення документа)

Моделі систем логічного управління на основі мікроконтролерних пристроїв

(тема)

Виконав: студент 2 курсу, групи СКСм-18-1

_____Гога М.В._____
(прізвище, ініціали)

Спеціальність *123_Комп'ютерна інженерія*

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

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(шифр і назва спеціальності, освітньої прграми)

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

Допускається до захисту

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

2019 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління_____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_Комп'ютерна інженерія_____
(шифр і назва)

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

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва

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

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові_____Гога Максиму Валерійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Моделі та методи верифікації автоматних систем логічного управління_____
_____
Logic Control System Models Based on Microcontroller Devices_____

затверджена наказом по університету від "_04_"__11__2019 р. № __1624 Ст_____.____

2. Термін подання студентом роботи до екзаменаційної комісії _____24.12.2019_____

3. Вихідні дані до роботи (проекту)_____
Мікроконтролер STM32F429ZIT6_____
Мова програмування Сі_____
Бібліотека HAL_____
Середовище розробки Keil for ARM_____
САПР STM32CubeMX_____
Логічний аналізатор для апаратного відлагодження_____
_____

4. Перелік питань, що потрібно опрацювати у роботі _____
Методи проектування систем логічного управління_____
Методи автоматного програмування при побудові моделей автоматів_____
Система обробки подій та переривань_____
Технологічна платформа реалізації_____
Автоматна модель пристрою керування_____
Середовище розробки програмного забезпечення_____
Програмування мікроконтролерного пристрою_____

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 03.09.2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Видача теми проекту, узгодження і затвердження	03.09.2019 -10.09.2019	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	10.09.2019 -30.09.2019	
3	Проектування систем логічного управління	30.09.2019 -15.10.2019	
4	Аналіз методу автоматного програмування на мікроконтролерних системах	15.10.2019 -15.11.2019	
5	Розробка сценаріїв програмної реалізації	15.11.2019 - 25.11.2019	
6	Розробка моделі пристрою керування	25.11.2019 -05.12.2019	
7	Розробка програми для мікроконтролера	05.12.2019 -15.12.2019	
8	Оформлення пояснювальної записки	15.12.2019 -20.12.2019	
9	Захист проекту	20.12.2019 -25.12.2019	

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка містить 86 сторінок, 30 рисунків, 1 таблицю, 20 джерел за переліком посилань.

ОСВІТЛЕННЯ, ПІШОХІДНИЙ ПЕРЕХІД, АВТОМАТНЕ ПРОГРАМУВАННЯ, ОБРОБКА ПОДІЙ, ГРАФ ПЕРЕХОДІВ, СИСТЕМИ ЛОГІЧНОГО УПРАВЛІННЯ, СИСТЕМИ РЕАЛЬНОГО ЧАСУ, ЧАСОВИЙ АВТОМАТ, АВТОМАТНИЙ ШАБЛОН, МІКРОКОНТРОЛЕР, STM32F429ZIT6, МОВА ПРОГРАМУВАННЯ C, IDE KEIL FOR ARM, САПР STM32CUBEMX.

Метою роботи є шаблон побудови опису моделей систем реального часу з використанням технології автоматного і подійного програмування, з найліпшим способом опису часових параметрів.

Розроблений прототип пристрою управління освітленням пішохідного переходу з використанням підсвічування смуг переходу. Для реалізації пристрою управління застосований часовий керуючий автомат Мура, математична модель якого представлена в формі темпорального графу переходів з використанням гіперпроцесу та обробника зовнішніх подій.

Програмна реалізація пристрою здійснена за технологією автоматного програмування. Побудова опису пристрою здійснена за концепцією автоматного шаблону, який відповідає критеріям систем логічного управління та систем реального часу.

Керуючий автомат реалізований на базі мікроконтролера STM32F429ZIT6. Системні налаштування здійснені за допомогою САПР STM32CubeMX. Програмна реалізація алгоритму управління здійснена на мові програмування C у середовищі розробки Keil for ARM.

ABSTRACT

Bachelor's thesis contains 85 pages, 30 figures, 1 table, 20 sources according to the list of links.

LIGHTING, PEDESTRIAN CROSSING, STATE MACHINE PROGRAMMING, EVENTS PROCESSING, STATE DIAGRAM, LOGIC CONTROL SYSTEMS, REAL-TIME SYSTEMS, REAL-TIME STATE MACHINE, STATE MACHINE TEMPLATE, MICROCONTROLLER, STM32F429ZIT6, C LANGUAGE, IDE KEIL FOR ARM, STM32CUBEMX CAD

The purpose of the work is to develop a template for constructing a description of real-time systems models using state machine and event programming technology, with the best way to describe time parameters.

A prototype of a pedestrian crossing lighting control unit was developed using the transition strip illumination. A Moore time controller is used to implement the control device, which mathematical model is presented in the form of a temporal state diagram using a hyperprocess and an external event handler.

The software implementation of the device is made using the technology of state machine programming. The description of the device is based on the concept of a state machine template that corresponds the criteria of logic control systems and real-time systems.

The control unit is implemented on the basis of the microcontroller STM32F429ZIT6. The system settings are made using the STM32CubeMX CAD software. The software implementation of the control algorithm is implemented in C programming language in the Keil for ARM development environment.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 АВТОМАТНІ СИСТЕМИ ЛОГІЧНОГО УПРАВЛІННЯ 12
1.1 Системи логічного управління 12
1.2 Кінцеві автомати або автоматні пристрої керування 14
1.3 Обробка подій у системах керування 19
1.4 Візуальний формалізм представлення моделей автоматів 25
1.5 Системи реального часу. Часові автомати 26
1.6 Гіперавтомати у системах керування 29
1.7 Модель пристрою керування освітлення пішохідних переходів 34
2 ПРИСТРОЇ КЕРУВАННЯ ОСВІТЛЕННЯМ ПІШОХІДНИХ ПЕРЕХОДІВ 38
2.1 Периферійне обладнання для освітлення пішохідних переходів 38
2.2 Технічне завдання на проектування 42
2.3 Принципи автоматного програмування 44
3 АВТОМАТНИЙ ШАБЛОН В РЕАЛІЗАЦІЇ СИСТЕМ ЛОГІЧНОГО УПРАВЛІННЯ 48
3.1 Апаратні та програмні компоненти реалізації пристрою керування. 48
3.2 Налаштування периферії МК 49
3.3 Опис стандартного автоматного шаблону для пристрою на базі мікроконтролеру 56
3.4 Альтернативний опис пристрою за концепцією автоматного шаблону 71
3.5 Результат роботи пристрою. Порівняння шаблонів 76
ВИСНОВКИ 82
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 84

ДОДАТОК А Графічна частина атестаційної роботи....**Ошибка! Закладка не определена.**
ДОДАТОК Б Програмна реалізація пристрою.....**Ошибка! Закладка не определена.**
Заголовний файл (власний код, файл main.h).....**Ошибка! Закладка не определена.**
Опис необхідних змінних, створення затримок та ініціація роботи керуючого автомату (власний код, main.c).....**Ошибка! Закладка не определена.**
Опис стандартного шаблону часового автомату (власний код, файл main.c) ..... **Ошибка! Закладка не определена.**
Опис альтернативного сценарію з реалізацією механізму розщеплення станів (власний код, файл main.c)..... **Ошибка! Закладка не определена.**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ДТП – дорожньо-транспортна пригода;

ОП – освітлювальні прилади;

ПВТ – пішохідне викличне табло;

СЛУ – система логічного управління;

ОУ – об’єкт управління;

СРЧ – система реального часу;

АП – автоматне програмування;

АШ – автоматній шаблон;

КА – керуючий автомат;

КП – керуючий пристрій;

ЦП – цифровий пристрій;

ПЗ – програмне забезпечення;

МК – мікроконтролер;

САПР – система автоматизованого проектування;

IDE – Integrated Development Environment (інтегроване середовище розробки);

HAL – Hardware Abstraction Layer (слой апаратних абстракцій);

GPIO – General-Purpose Input/Output (Інтерфейс введення/виведення загального призначення)

NVIC – Nested Vectored Interrupt Controller (контролер вкладених векторів-переривань);

IVT – Interrupt Vector Table (таблиця векторів-переривань);

IRQ – Interrupt ReQuest (переривання, запит на переривання);

ISR – Interrupt Service Routine (обробник переривань).

ВСТУП

На основі загальної концепції побудови систем автоматизованого управління в них завжди можна виділити пристрої управління та об'єкти управління. Дотримуючись цієї концепції, системи управління на основі кінцевих автоматів (finite state machines) також можна розділити на дві частини:

– керуючу частину, відповідальну за логіку поведінки - вибір виконуваних дій, що залежить від поточного стану і вхідного впливу, а також за перехід в новий стан;

– керовану частину, відповідальну за виконання дій, обраних для виконання керуючою частиною, і, можливо, за формування деяких компонентів вхідних впливів для керуючої частини - зворотних зв'язків.

Серед усієї множини керуючих пристроїв можна виділити пристрої логічного управління, у яких керуючі сигнали (control value) представляються в двійковому алфавіті. Для реалізації керуючої частини в таких пристроях, як правило, використовуються кінцеві керуючі автомати (КА). Подібні пристрої широко застосовуються в системах Internet of Things.

Будь-який локальний цифровий пристрій, що реалізує алгоритм обробки інформації або управління, може бути реалізовано двома способами: апаратним або програмно-апаратним.

При програмно-апаратному способі реалізації алгоритм описується на апаратно-орієнтованій мові програмування (наприклад, на мові Сі зі спеціальними бібліотеками) з урахуванням апаратної архітектури, на якій буде реалізовуватися задана програма. Це, як правило, різні сімейства мікроконтролерів (МК). Перевагою даного підходу є наявність спеціальних апаратно-орієнтованих функцій (таймерів контролерів переривань), а також наявність апаратно реалізованих інтерфейсів обміну із зовнішніми пристроями. До недоліків можна віднести меншу швидкодію і обмеження, що

накладаються апаратної архітектурою на реалізований алгоритм.

При описі алгоритму функціонування цифрових пристроїв логічного керування в САПР цифрових пристроїв одним із стилів написання коду є стиль автоматного програмування. Суть автоматного програмування полягає у розділенні опису логіки поведінки (за яких умов необхідно виконати ті чи інші дії) від опису його семантики (власне сенсу кожного з дій). У автоматному програмуванні, в якості базового, використовується поняття «стан», а не «клас», «об'єкт», та ін. А в якості опису алгоритму функціонування використовується граф переходів.

Автоматні програми строго структуровані і в них виділено три види функцій: функції переходів, функції виходів, функції реалізації затримок і переходу в новий стан. З огляду на напрямок розвитку у сфері систем реального часу, необхідно відзначити те, що також існує узгодження затримок, які відповідають за логіку роботи пристрою в певний момент часу.

Автоматні програми строго шаблонізовані, з використанням операторів багатопозиційного вибору (switch, case), умовних операторів (if, select) і функцій реалізації таймера або фронту (синхросигналу Clk). Автоматні програми інваріантні до способу кодування (мови програмної реалізації). Є приклади автоматних програм на різних мовах опису апаратури, на C, на JScript та ін.

Таким чином, метою роботи є шаблон побудови опису моделей систем реального часу з використанням технології автоматного та подійного програмування, з найліпшим способом опису часових параметрів.

Об'єктом дослідження є моделі систем логічного управління на основі кінцевих автоматів.

Предметом дослідження є способи опису моделей систем логічного управління на основі мікроконтролерних пристроїв на мові програмування CI.

Наукова новизна роботи полягає у аналізі способів опису моделей систем логічного управління на основі мікроконтролерних пристроїв на мові програмування Сі, а саме: стандартного опису моделей, опису моделей за

технологією автоматного програмування з використанням часового автоматного шаблону зі всіма часовими параметрами (затримка вихідних значень, затримка станів та вікно обмежень) та опису моделей з використанням технології розщеплення станів. Всі розглянуті способи опису моделей підтримують найліпший спосіб опису часових параметрів та принцип подійного програмування.

Практична цінність роботи полягає в можливості апаратної реалізації систем реального часу на основі мікроконтролерних пристроїв, що дає можливість збільшити гнучкість розроблюваних систем, тобто розширити технічні та технологічні можливості при розробці даних систем.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз методів проектування систем логічного управління;

– провести аналіз методів проектування систем реального часу;

– провести аналіз методів автоматного програмування при побудові моделей часових автоматів на базі мікроконтролера;

– провести аналіз принципу побудови системи обробки подій та переривання;

– створити програмну реалізацію пристрою за всіма характеристиками систем логічного управління та систем реального часу;

– проаналізувати недоліки та переваги створеної програмної реалізації.

1 АВТОМАТНІ СИСТЕМИ ЛОГІЧНОГО УПРАВЛІННЯ

1.1 Системи логічного управління

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

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

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

Інформація про роботу ОУ надходить до СЛУ від сигналізаторів положення, стану і параметрів (датчиків та сенсорів). При цьому необхідно зазначити, що сигналізатори параметрів складаються з двох складових – датчика і вторинного приладу (перетворювача), який здійснює порівняння значення аналогового сигналу від датчика із заданими значеннями параметрів, і формування двійкового сигналу {0, 1}. При цьому «0» визначається виходом значення за межі встановлених діапазонів, а «1» знаходженням значення параметру у заданому діапазоні [2].

Загальна структура системи автоматичного управління показана на рис.1.

В основі моделей систем логічного управління лежить поняття «стану». Стан це математична абстракція, яка однозначно ставиться у відповідність кожному з фізичних станів об'єкта управління, 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 [3].

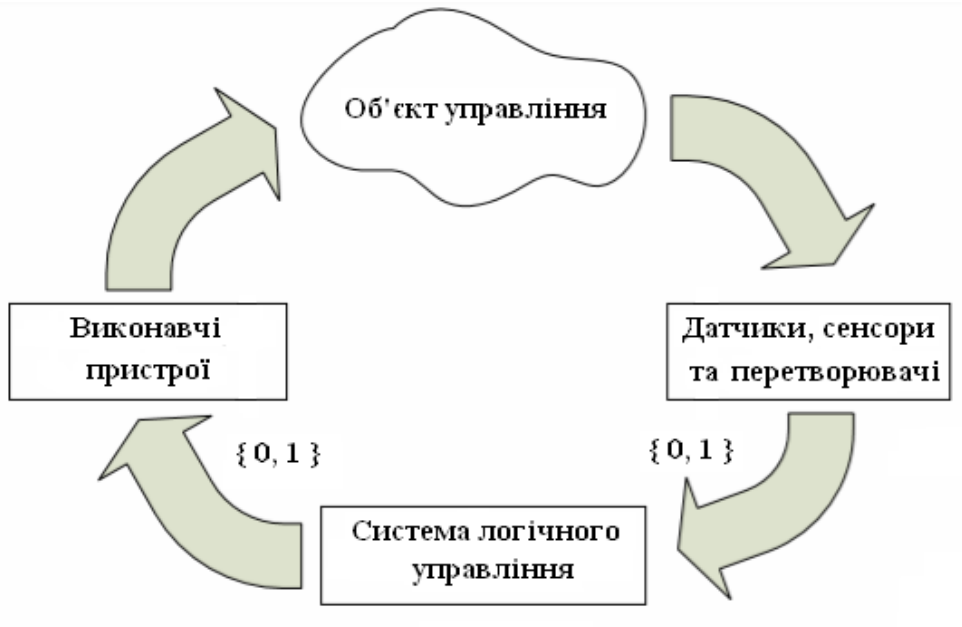

Рисунок 1.1 – Загальна структура системи автоматичного управління

Стан – це сукупність параметрів технічної системи в даний момент часу. Поточний стан несе в собі усю інформацію про минуле системи, необхідну для визначення її реакції на будь-яку вхідну дію, що формується у даний момент часу. Стан можна розглядати як особливу характеристику, яка в неявній формі об'єднує усі вхідні дії минулого, що впливають на реакцію системи. Реакція системи у цій термінології залежить тільки від вхідної дії і поточного стану. Таким чином стан можна визначити як сукупність параметрів моделі динамічної технічної системи, значення яких залежать від вхідних сигналів та передісторії даних параметрів. Кожен стан технічної системи, окрім сукупності своїх параметрів, характеризується часом знаходження системи у зазначеному стані (Т). Перейти у новий стан система може по закінченню часу Т (при наявності відповідних значень вхідних сигналів) або за подією, яка безпосередньо змінює стан технічної системи.

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

1.2 Кінцеві автомати або автоматні пристрої керування

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 [4]. Абстрактний кінцевий автомат представляється шестіркою $W =< X, A, Y, \delta, \lambda, a_0 >$, де $X = \{x_1, x_2, ..., x_m\}$ – множина букв вхідного алфавіту; $A = \{a_1, a_2, ..., a_n\}$ – множина станів автомату; $Y = \{y_1, y_2, ..., y_r\}$ –множина букв вихідного алфавіту; $\delta(a_i, x_k) = a_j$ – функція переходів автомату, $\lambda(a_i, x_k) = y_\alpha$ – функція виходів автомату, a_0 – початковий стан автомату.

Структурні моделі автоматів, які використовуються в задачах логічного управління, можуть розглядатися як пристрої керування. Тому в усіх структурних моделях існує множина вхідних сигналів X, множина станів A і множина вихідних реакцій Y (усі три множини є кінцевими). У завданнях логічного управління значення має не лише кінцева множина X, A і Y, але і їх точна розмірність, оскільки вона впливає на складність реалізації пристрою керування. Символам вхідного та вихідного алфавітів ставляться у відповідність вектори вхідних (X) та вихідних (Y) змінних, а при схемній реалізації – зовнішніх входів та виходів схеми. Кожен стан абстрактного автомата a_i кодується вектором внутрішніх змінних $Z_i = \{z_{i,1}, z_{i,2}, ..., z_{i,k}\}$.

Поняття «значення вхідних сигналів» (input values), поділяється на

вхідні дії (input actions) та події (events). 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), в свою чергу поділяється на «вихідні дії» (output actions) і «вихідну діяльність» (output activity), Передбачається, що дія є одноразовою, миттєвою і неперервною, а діяльність може тривати досить довго і можливо її переривання деякою вхідною подією [3].

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

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

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

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

Відповідно до традиційної теорії автоматичного управління, керована частина визначається як об'єкт управління, а частина, що керує як система управління. Оскільки для реалізації системи управління використовуються автомати, то вона часто називається керуючий автомат або просто автомат. Сукупність об’єкту управління та керуючого автомату прийнято визначати як автоматну систему управління.

На рис.1.2 наведена структура автоматизованої (автоматної) системи логічного управління, де Y – множина керуючих сигналів, X_C – множина

оповіщувальних сигналів від об’єкта управління, X_E – множина зовнішніх сигналів (подій).

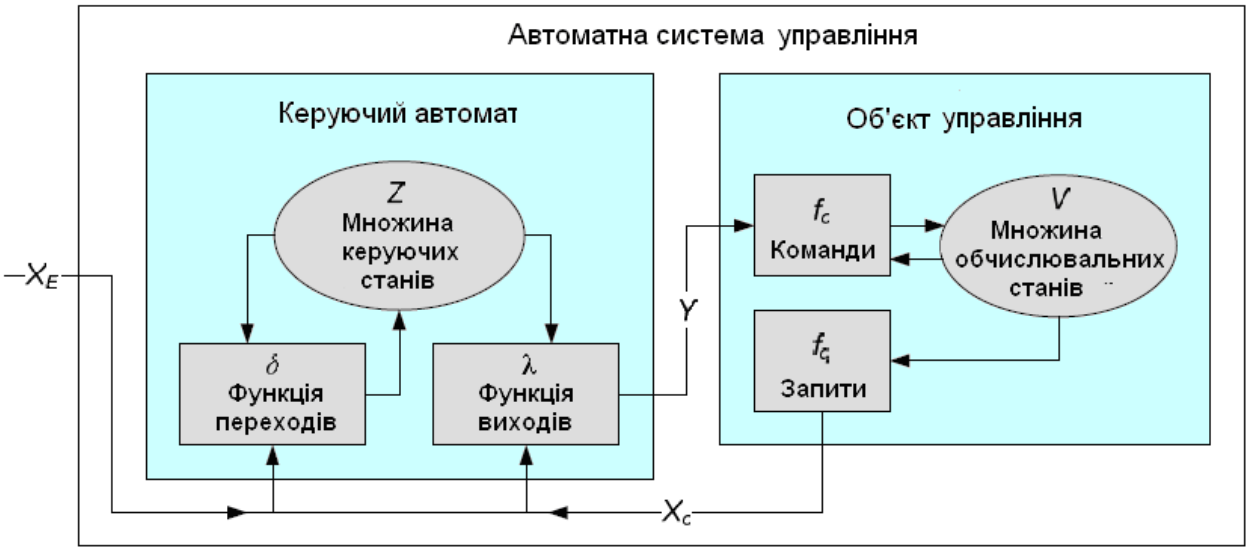

Рисунок 1.2 – Структура автоматної системи логічного управління

Важливою характеристикою системи логічного управління є її кінцевість. Керуючий автомат не тільки має кінцеве число станів, а й реалізовані їм функції переходів та виходів оперують виключно кінцевими множинами сигналів. Саме ця властивість дозволяє описувати логіку поведінки автоматів явно у вигляді таблиць або графів переходів. Слід зазначити, що існує неформальна вимога: число керуючих станів, вхідних і вихідних сигналів має бути невеликим і доступним для огляду. Керуючий автомат з сотнею станів, безумовно, є кінцевим, однак, зобразити його граф переходів практично неможливо, що зводить нанівець переваги явного виділення керуючих станів.

Навпаки, число обчислювальних станів може бути як завгодно великим. В процесі роботи керуючому автомату потрібно отримувати інформацію про обчислювальні стани і змінювати їх. Однак в силу властивості кінцевості автомат не може безпосередньо зчитувати і записувати обчислювальний стан. Для цього і потрібні операції об'єкта керування.

Розглянемо модель часу в системах логічного управління. Час може вважатися таким, що тече безперервно або дискретно. У першому випадку час змінюється на континуумі і має назву метричного або реального часу, в другому – на перерахованій множині, елементи якої називаються тактами t_i, і має назву дискретного або автоматного часу. Системи, в яких час дискретний, число узагальнених координат (змінних) кінцеве і кожна координата може набувати значень з кінцевої множини (алфавіту), називають кінцевими динамічними, тобто такими, що змінюються в часі, системами. Кінцеві автомати належать до класу кінцевих динамічних систем [5].

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

Враховуючи, що система логічного управління може знаходитися у зазначеному технічному стані визначений проміжок реального часу T, тривалість знаходження у кожному автоматному стані може бути один або декілька автоматних тактів, виходячи з тривалості автоматного такту у реальному часі. Початок кожного автоматного такту визначається виключно подією. Ця подія може бути зовнішньою по відношенню до автомату (як і до всієї СЛУ), а може бути внутрішньою. Слід зазначити, що при апаратній реалізації синхронних структурних автоматів початок кожного автоматного такту ініціюється тактовим генератором, який задає синхропослідовність Clk. Таким чином, синхропослідовність є системою внутрішніх подій синхронного автомату. Синхропослідовність характеризується частотою $f = 1/\tau$, де τ – період часу між однаковими подіями синхропослідовності (фронтами).

Для стабільної роботи автомата в циклі керування визначимо обмеження, що опитування значень вхідних змінних (які відповідають діям) здійснюється один раз за період автоматного такту, і в одному автоматному

такті обробляється одна подія (або декілька невиключних подій).

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

Якщо в одному автоматі використовуються обидва способи формування значень вихідних змінних, то він називається «змішаним» (С-автоматом (СА)). З цієї точки зору більшість структурних автоматів є змішаними. Якщо в ньому одні й ті ж змінні застосовуються як в якості вхідних, так і вихідних змінних, то такий автомат називається «автоматом з прапорами (флагами)».

Таким чином, апаратна реалізація моделі структурного автомата (модель Хаффмена), складається з комбінаційного і послідовностного компонентів. Послідовностний компонент містить елементи пам'яті, такі як синхронні тригери, які запам'ятовують значення внутрішніх змінних (стан) і дозволяють змінювати його синхронно. Комбінаційний компонент складається з логічних елементів, які реалізують дві логічні функції: функцію виходів, яка обчислює значення вихідних сигналів, і функцію переходів, яка обчислює нові значення елементів пам'яті або внутрішніх змінних (тобто значення наступного стану) [6]. На рис.1.3 наведена графічна інтерпретація апаратної реалізації структурного автомату.

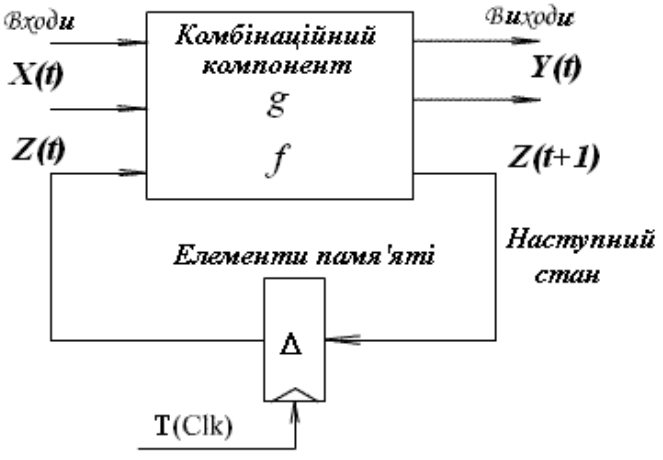

Рисунок 1.3 – Модель структурного автомату

Надалі (якщо не обумовлено інше) будемо розглядати автомати, робота яких тактується сигналами від генератора синхронізуючих імпульсів (CLK), тобто в автоматі крім входів (x_1,..., x_n) є принаймні ще один вхід CLK, за яким надходить синхронізуюча послідовність. Тому вхідним сигналом на переході (a_m, a_s), відповідним шляху $a_m X$ (a_m, a_s) – Y (a_m, a_s) a_s, буде не X (a_m, a_s), а кон'юнкція CLK · X (a_m, a_s). Таким чином, при X (a_m, a_s) = 1 (безумовний перехід) автомат працює тільки від синхроімпульсу. Але при побудові графа переходів автомата символ CLK дугам графа переходів, як правило, не приписується.

1.3 Обробка подій у системах керування

У багатьох системах логічного управління керування здійснюється виключно зовнішніми подіями. Розглянемо принципи обробки подій, які достатньо докладно розроблені в алгоритмічних мовах програмування [7].

Подія – це абстрактне поняття, яке має на увазі таку зміну зовнішнього середовища, яка має на увазі певну реакцію системи.

У програмних системах в загальному випадку подіями можуть бути:

– графічна взаємодія користувача з комп'ютером (за допомогою миші, електронної кисті, жестів на сенсорному екрані);

– завершення операцій введення-виводу;

– події в Інтернеті (завантаження веб-сайту, його реакція);

– розпізнавання образів в зображеннях;

– реакції інтелектуальних будинків або сучасних автомобілів на поточну обстановку;

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

Обробка подій подібна до обробки виключень. У обох випадках задіються обробники. Хоча виключення можуть бути згенеровані або явно

програмним кодом, або неявно апаратними засобами, події створюються зовнішніми діями. Вважатимемо, що більшість подій викликана взаємодією користувача з елементами графічного інтерфейсу користувача (GUI), наприклад кнопками. Реакція на такі взаємодії з користувачем є найбільш поширеною формою обробки подій.

Широке поширення подієво-орієнтованого програмування обумовлене саме популярністю графічних інтерфейсів. Багато веб-документів зараз динамічні. Такий документ може надавати, наприклад, форму замовлення для користувача, який вибирає товар, клацаючи по кнопках. Необхідні внутрішні обчислення, пов'язані з цими клацаннями, виконуються обробниками подій, що реагують на події клацання. Ще одне поширене використання обробників подій - перевірка правильності заповнення форм, коли вони міняються або коли форма передається на веб-сервер для обробки.

У програмуванні на основі подій виділяють п'ять елементів:

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

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

Слухач події приймає сигнал і передає його характеристики (як об'єкт події) через адаптер в потрібний обробник події, що виконує дії (залежно від стану системи). Якщо слухач обслуговує декілька адаптерів, він додатково

вирішує задачу диспетчера (комутує шлях сигналу). Обробка події ускладнюється, коли декілька подій спільно використовують загальний ресурс. В цьому випадку подіям призначаються різні пріоритети (але часу, значущості і т. д.).

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

У контур обробки події може включатися адаптер. Адаптер події це інтерфейс із порожніми методами, які відповідають подіям. Порожні методи перевизначаються конкретними класами обробників.

Говорять, що адаптер події підключає слухача подій до обробника подій. Адаптери подій дуже корисні, коли для одного джерела подій є декілька споріднених методів обробки. (наприклад, миша є джерелом подій і з нею пов'язано декілька схожих подій: клацання мишею, переміщення миші, відпуск миші і т.д.). Після того, як подія миші активована, обробник миші, що реалізує інтерфейс адаптера миші, перевизначає відповідний порожній метод.

Взаємодію елементів моделі на основі подій показано на рис.1.4. В основі моделі знаходиться джерело подій. Слухач зайнятий прослуховуванням джерела подій. Після виявлення згенерованого сигналу слухач події відправляє дані в окремий обробник подій (через адаптер подій). Залежно від поточного стану системи обробник події виконує необхідні дії над прикладним об'єктом, і система переходить в новий стан.

У деяких мовах програмування ролі учасників розглянутої подієвої моделі поєднуються (наприклад, роль слухача і адаптера).

У додатках з графічним інтерфейсом користувача досить багато потенційних подій. Проте подія відкидається, якщо для цього не передбачений обробник.

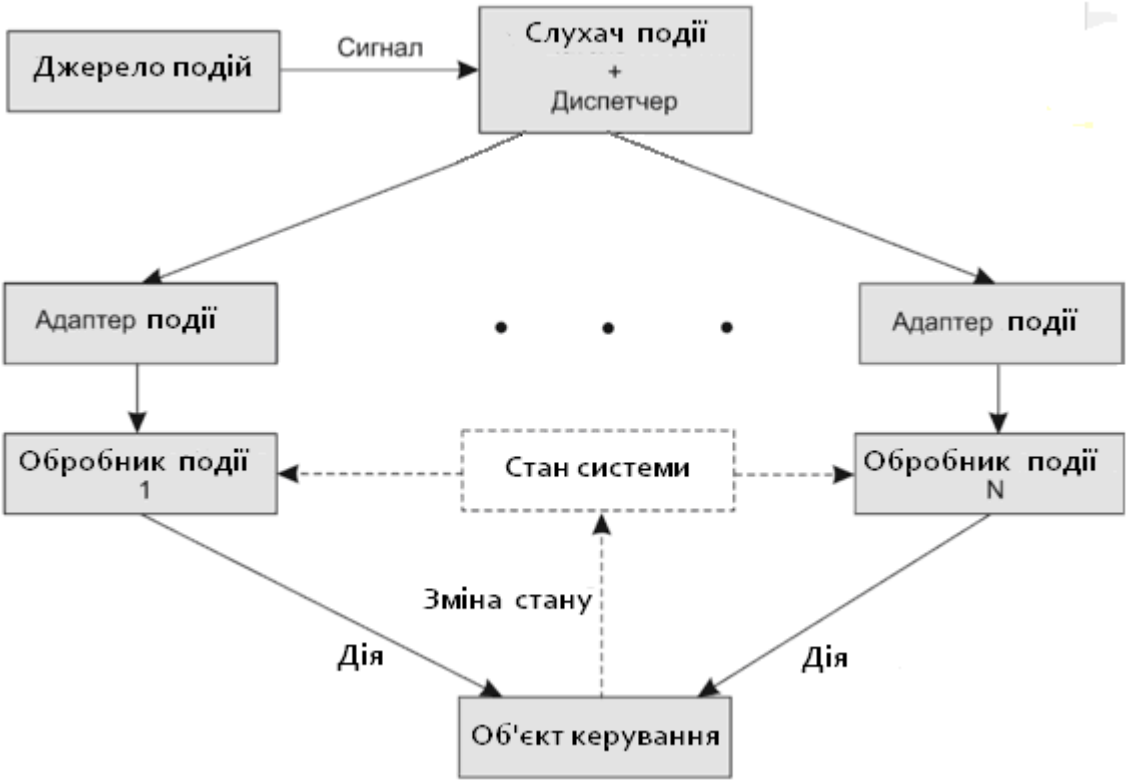

Рисунок 1.4 – Модель програмування на основі подій

Обробка подій, як правило, пов'язана з системами реального часу, які визначаються, виходячи з динамічних характеристик контрольованих процесів і пов'язаних з ним подій. При цьому під подією розуміється така зміна середовища, яке має на увазі певну реакцію системи. Наприклад, зміна атмосферного тиску є подією для барометра, але не є подією для секундоміра. Подія – абстрактне поняття: події можуть генеруватися як зовнішнім середовищем. так і всередині самої системи се компонентами. Приклад зовнішнього події - натискання на кнопку. Затримка, з якою система повинна обробити це натискання, і стане вихідною вимогою реального часу. У свою чергу, і механічний вплив на кнопку призводить до виникнення вторинних внутрішніх подій – зміни напруги на одному з входів мікроконтролера.

Одні події можуть породжувати інші події більш високого рівня абстракції. Наприклад, в результаті обробки низькорівневої події – зміни напруги в коді кнопки, блоком придушення брязкоту генеруються події середнього рівня – натиснення і відпуск кнопки. У свою чергу, на їх основі

можна згенерувати високорівневі події – подвійне натискання або натискання з повтором, якщо кнопка утримується тривалий час.

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

Типова система управління одночасно обробляє події в умовах «жорсткого» та «м'якого» реального часу. Події першої групи зазвичай пов'язані з контрольованим об'єктом, цифровою обробкою або синтезом сигналів і прив'язані до синхросигналом часової сітки (clock driven). Події другої групи пов'язані з призначеним для користувача або міжсистемним інтерфейсом і мають асинхронну, випадкову природу (event-driven). Відповідні функції обробки об'єднують в синхронну і асинхронну підсистеми.

Час реакції системи на будь-яку подію визначається затримкою фіксації і тривалістю його обробки. Ці параметри, в свою чергу, залежать від вибраних алгоритмів і їх реалізації.

Фіксація події – це зміна ходу обчислювального процесу і/або зміна параметрів обчислень, викликані цією подією. Виділяють наступні способи фіксації:

– опитування (polling) – передбачає читання у виконуваній програмі даних з порту введення-виведення (загальною осередки пам'яті) і виявлення змін, викликаних зовнішнім подією, за відсутності події процедура опитування виконується «вхолосту»;

– переривання (interrupt) - ініціюється зовнішньою подією поточних обчислень і негайний запуск підпрограми – обробника переривання (interrupt service routine, ISR);

– комбінований, наприклад прямий, доступ до пам'яті (direct memory access, DMA).

Класичне застосування опитування – циклічне введення даних, обробка і виведення. Зазвичай період такого циклу задається вбудованим таймером, завершення роботи якого також фіксується методом опитування.

Окремим випадком систем з обробкою переривань є так звані foreground/background-системи, що включають основну програму типу «петля» (background) і обробники переривань (foreground) (рис.1.5).

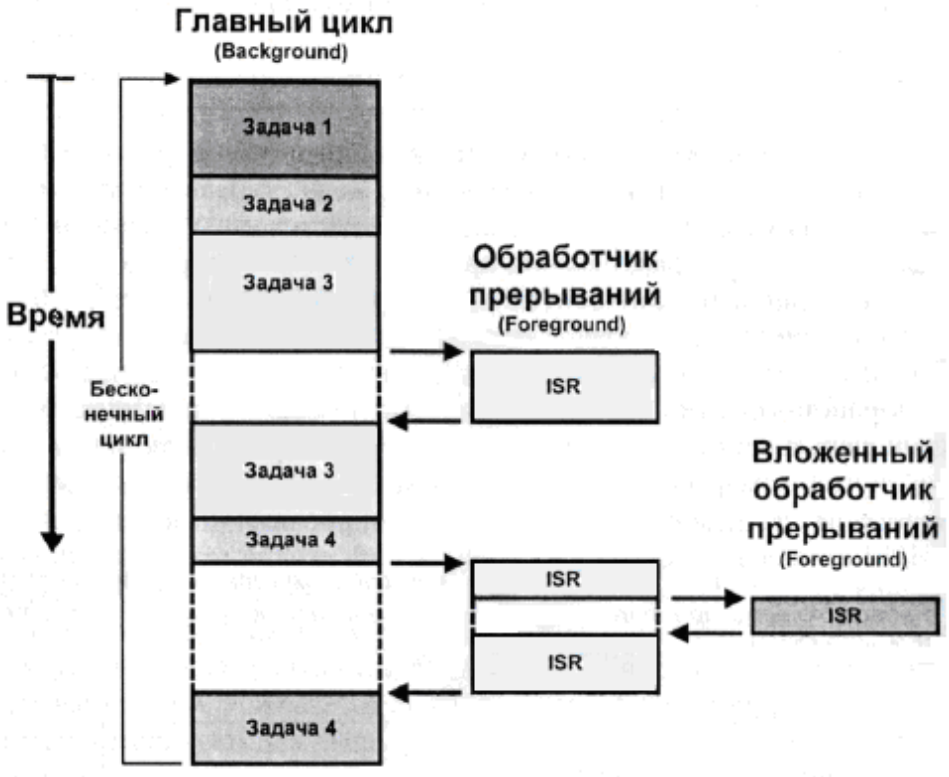

Рисунок 1.5 – Виконання програми з вкладеністю переривань

Оскільки обробка переривань реалізована апаратно, такі системи вимагають мінімальних витрат для організації багатозадачності. Взаємне виключення зазвичай реалізується шляхом тимчасового блокування (маскування) переривань, а для збереження контексту при виклику обробників

використовується єдиний стек основної програми, ємність якого можливо збільшити з урахуванням наявної вкладеності переривань [8].

1.4 Візуальний формалізм представлення моделей автоматів

В якості алгоритмічної моделі для керуючих автоматів застосовується такий непроцедурний візуальний формалізм, як графи переходів, які в літературі також називаються діаграмами станів (State Diagrams). У цьому підході, подія є вторинна і разом з вхідною змінною розглядається як різновид вхідної дії, яка може змінити стан.

Одне з достоїнств графів переходів, які мають бути "максимально" пленарними, полягає в тому, що на кожній його дузі вказуються не усі вхідні сигнали, а тільки ті з них, які забезпечують перехід по цій дузі. На кожній дузі вхідні сигнали можуть об'єднуватися в булеві формули, у тому числі і довільної глибини, що різко підвищує компактність опису алгоритму. Введення в модель булевих формул забезпечує "паралельну" обробку вхідних дій. Можна стверджувати, що кожен стан в графі переходів "виділяє" з множини усіх вхідних дій тільки ту їх підмножину, яка забезпечує переходи з цього стану. Це дозволяє застосовувати графи переходів для вирішення задач логічного управління при великому числі вхідних дій, значно спрощуючи вирішення проблеми тестування по усіх шляхах.

Забезпечення "паралелізму" по виходах досягається за рахунок вказування в компонентах (у вершинах і/або на дугах) графа переходів значень вихідних змінних (діяльностей), сформованих в цих компонентах. При цьому безпосереднє вказування в кожній компоненті (залежно від вибраної структурної моделі автомата Милі або Мура) значень усіх вихідних змінних різко спрощує розуміння алгоритму, описаного графом переходів, Побудований таким чином граф переходів задає структурний кінцевий автомат без замовчань значень вихідних змінних. На рис.1.6 наведений фрагмент графу переходів простого автомату, який не враховує параметрів

часу.

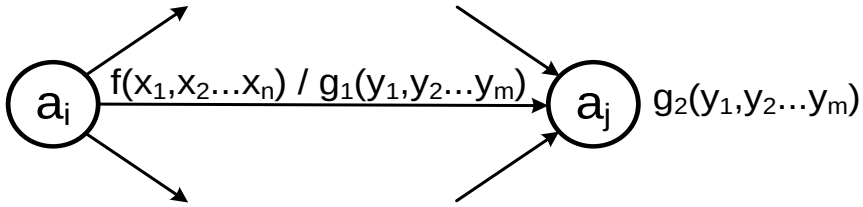

Рисунок 1.6 – Фрагмент простого графа переходів структурного автомата

На рис.1.6 $f\{x_1, x_2,..., x_m\}$ – логічний вираз для функції переходів (a_i –> a_j), та $\{x_1, x_2,..., x_m\}$ – набір умов (значень вхідних змінних), що ініціюють перехід (a_i —> a_j);

$g_2\{y_1, y_2,..., y_m\}$ – функція виходів (вихідна дія або діяльність, логічний вираз або функція) на переході (a_i —> a_j) в моделі Мілі;

$g_2\{y_1, y_2,..., y_m\}$ – функція виходів (вихідна дія або діяльність, логічний вираз або функція) для стану a_j в моделі Мура,

$\{y_1, y_2,..., y_m\}$ – набір реакцій автомата (значень вихідних змінних).

В даній роботі розглядається модель Мура.

Варто зазначити, що кожен граф переходів має бути семантично і синтаксично коректний. Перша властивість визначає, чи правильна побудована модель, а друга – чи правильно вона побудована.

В даній роботі граф семантично та синтаксично коректний, але простий візуальний формалізм у вигляді графу переходів в даному проекті не дозволяє детальніше розібрати всі критерії правильності відтворення графу переходів.

1.5 Системи реального часу. Часові автомати

Система управління реального часу (СРЧ) – система, в якій результуюча дія (діяльність) залежить не лише від логічних значень простих управляючих дій, але і від часу, протягом якого ці дії виконуються. Головна відмінність завдань в реальному часі від завдань, не залежних від часу, полягає в тому, що

в системах реального часу завдання має завершитися в межах заданого проміжку часу, який дозволяє коректно завершити процес обробки даних.

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) [9-11]. Часова змінна (timed variable) постійно збільшує своє значення і "скидається" в 0 при приході вхідного сигналу (як правило, це подія), або при видачі вихідного сигналу. Часові змінні вимірюються в автоматних тактах.

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

Якщо кожну вершину графа переходів помітити вектором, що складається з необхідних значень всіх вихідних змінних, то процес буде таким, що спостерігається за виходами, а значення кожної вихідної змінної не буде залежати від передісторії. При цьому тривалість формування векторів значень вихідних змінних визначається часом перебування в вершині, тобто часом активності вершини, що збігається з часом виконання умови, яка помічає петлю. В цьому випадку суміжні вершини при перегляді в одному напрямку можуть бути пов'язані тільки однією дугою, поміченої однієї булевої формулою, а кожна вершина може мати тільки одну петлю, також позначену

однією такою формулою. Поряд з вхідними та вихідними змінними, які забезпечують взаємодію керуючого автомата з ОУ і входять до відповідного вектору, що належить кожній вершині (дузі)), до цього можуть входити також і часові змінні, які є для керуючого автомата внутрішніми.

Таким чином, повна модель структурного часового автомату може бути представлена дев'яткою $W = (X, Y, Z, f, g, z_0, T_c, T_{to}, T_d)$, де:

$X = \{ X_C, X_E \}$ – множина вхідних змінних, X_C – множина оповіщувальних сигналів від об'єкта керування, X_E – множина зовнішніх подій;

$Y = \{ Y_C, Y_F \}$ – множина вихідних змінних, Y_C – множина реакцій (керуючих сигналів), Y_F – множина діяльностей (вихідних функцій);

Z – множина внутрішніх змінних, які визначають кодування станів автомату;

f – функція переходів, g – функція виходів;

z_0 – код початкового стану автомата;

$T_c = \{ t_{c1}, t_{c2} ... t_{cp} \}$ – множина часових змінних для 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.

На рис.1.7 зображений фрагмент графа переходів часового автомата.

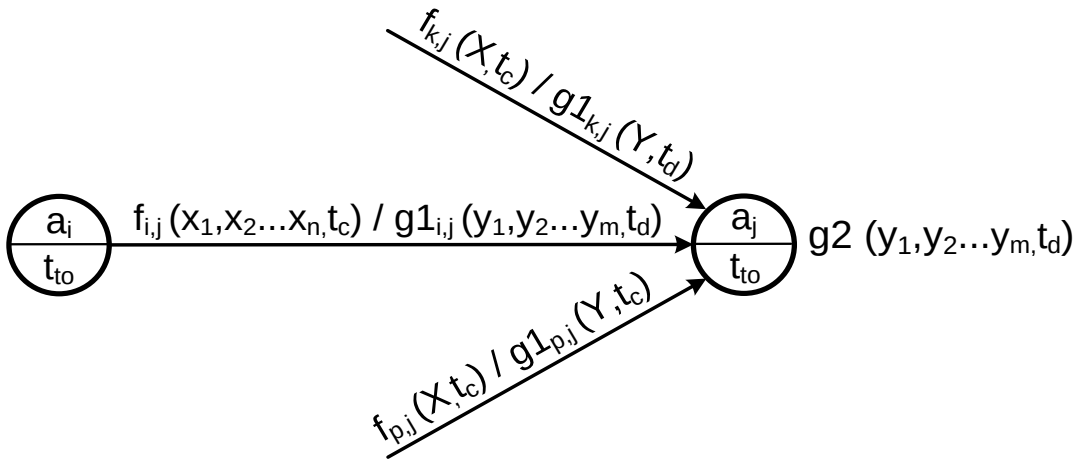

Рисунок 1.7 – Фрагмент графа переходів часового автомата

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

1.6 Гіперавтомати у системах керування

Нині практично уся промислова автоматизація реалізується на цифрових системах управління. Як базовий елемент систем управління використовуються програмовані логічні контролери (ПЛК). ПЛК функціонують відповідно до заданого алгоритму керування (АК).

Створення програм для ПЛК можливо тільки за наявності формальних мов і методик, самоіснування яких припускає серйозний теоретичний базис (модель керуючого автомату, понятійний апарат, термінологія), що відбиває наступну специфіку завдань автоматизації:

- гомеостаз, або наявність зовнішнього по відношенню до алгоритму середовища – об'єкта керування (ОК);
- циклічність функціонування АК;
- властиву завданню управління систему подій;
- необхідність синхронізації функціонування АК з фізичними

процесами на ОУ (служба часу);

- паралелізм фізичних процесів на ОК.

Попередній аналіз проблеми показує, що як основа моделі ОК може бути використана модель кінцевого автомата (КА). Проте пряме використання КА для моделювання складних ОК неможливо, оскільки, з одного боку, її привабливі властивості (можливість відбити гомеостаз, подійна система і циклічність) явно не підкреслені, а з іншого боку, простий КА не припускає операцій з часовими інтервалами і паралелізм у системах реального часу.

Для усунення цього протиріччя запропонована формальна модель складного алгоритму управління у вигляді гіперавтомата [12], яка задається п'ятіркою, $A = (F, x, y, f_{сиг}, f_1)$, і визначимо поняття про функції-стани. У цій моделі ми визначили $A = (F, x, y, f_{сиг}, f_1)$, як множину альтернативних функцій-станів, x, що відображують x в y ($y = fo_к(x), fo_к(x) \in O$) та x в F ($f_{cur}=f_{Sк}(x), f_{Sк}(x) \in F$). Зауважимо наступне. Немає причин так жорстко прив'язувати функції-стани до унікальних змінних, як це робиться в моделі кінцевого автомата. Змінні для кожної функції-стану можуть бути унікальними. Таким чином, змінні можуть прив'язуватися виключно до функції-стану. При цьому функція-стан може трактуватися виключно в сенсі програміста як звичайна функція або процедура (функція, що працює з глобальними змінними). Збережемо найменування "функція-стан" для віддзеркалення факту їх альтернативності, перемикання. Також як можливі функції-стани розглядатимемо і "порожні" функції, які не виконують ніяких дій. Такі "порожні" функції, значення яких обговоримо трохи пізніше, ми називатимемо "пасивними", а усі інші - "активними".

Зроблені зауваження дозволяють ввести поняття "процесу". Процес – це поліморфна функція, сукупність альтернативних функций-станів, що представляються в програмі як єдина неділима суть. При виклику цієї поліморфної функції (процесу) виконується одна і тільки один із складових його функцій-станів – поточна функція-стан. Додатково до цього кожен процес забезпечений індивідуальним «годинником» – лічильником часу, який

працює постійно і обнуляється при зміні поточної функції-стану.

Таким чином, процес p_i задається четвіркою елементів, які відбивають статичну і динамічну інформацію про процес, : $p_i = (F_i, f_i^1, fi^{cur}, T_i^{P,})$, де F_i - безліч функцій-станів процесу ($f_i^1 \in F_i$; fi^{cur} - поточна функція-стан, $fi^{cur} \in F_i$; T_i^P - поточний час знаходження процесу в поточній функції-стані, або поточний час відсутності переходів.

Статика процесу визначається елементами F_i і f_i^1, для розгляду процесу в динаміці додатково використовуються елементи fi^{cur} і T_i^P.

У свою чергу, *j*-а функція-стан довільного *i*-го процесу задається парою елементів: $f_i^j = (X_I{}^j, Y_I{}^j)$, де $X_I{}^j = \{ x_i{}^{j1}, \ldots , x_i{}^{1L}\}$ - множина подій, $f_i{}^j$; $Y_I{}^j = \{y_i{}^{j1}, \ldots , y_i{}^{1L}\}$ - множина реакцій f_{ij} .

Як подія функції-стану може розглядатися довільна суперпозиція фактів: значення поточної функції-стану деякого процесу, деяке певне значення змінної і деяке певне значення поточного часу відсутності переходів процесу (оператор timeout).

Реакцією називається довільна суперпозиція дій із зміни значень змінних і поточних функцій-станів процесів, визначувана на основі подій поточної функції. Серед функцій-станів процесу F_i розрізняються взаємно безліч активних і пасивних функцій-станів, що не перетинаються, F_{ia} та F_{ip} відповідно. Функція-стан процесу пасивна, якщо множина його реакцій порожня: $f_i^j \in F_i{}^p : X_i^j = \{\varnothing\}$.

Серед пасивних функцій-станів виділені функції f^{NS} "нормальний останов" і f^{ES} " помилковий останов ", які однакові для усіх процесів.

Сукупність спільно функціонуючих процесів утворює гіперпроцес (гіперавтомат) – надбудову над (гіпер) процесами. Модель алгоритму, що управляє, у вигляді гіперпроцесу термінологічно орієнтована на програмну реалізацію і дозволяє відбити перераховані властивості алгоритмів управління: відкритість, залежність від подій, циклічність, синхронізм і логічний паралелізм.

Гіперпроцес H є впорядкованим набором процесів, що циклічно

активізуються з періодом активізації T_H: $H = (T_H, P, p_1)$), де T_H - період активізації гіперпроцесу; P - множина процесів, $(P = \{p_1, p_2, ..., p_M\}$, де M - кількість процесів гіперпроцесу); p_1 - початковий процес, $p_1 \in P$.

По запуску гіперпроцесу поточна функція-стан виділеного процесу p_1 - початковий стан, для усіх інших процесів поточний стан – стан нормального останову: $f_1^{cur} = f_1^1, \ \forall i \neq 1 => f_1^{cur} = f^{NS}$. Пасивні функції служать для організації взаємодії між процесами і початкової ініціалізації.

Циклічна схема функціонування гіперпроцесу (рис. 1.8) подібна до схеми виконання кінцевого автомата, і якщо не враховувати той факт, що процеси слабкоприв'язані (впливають один на одного), то єдині відмінності - це множинність незалежних блоків і наявність тактування запуску циклів активізації (з періодом T_H). Строго кажучи, питання про те, як же активізуються процеси - фізично паралельно або ж послідовно, відкрите, але для простоти сприйняття можна вважати, що вони активізується послідовно, в порядку опису, як і показано на рис.1.8.

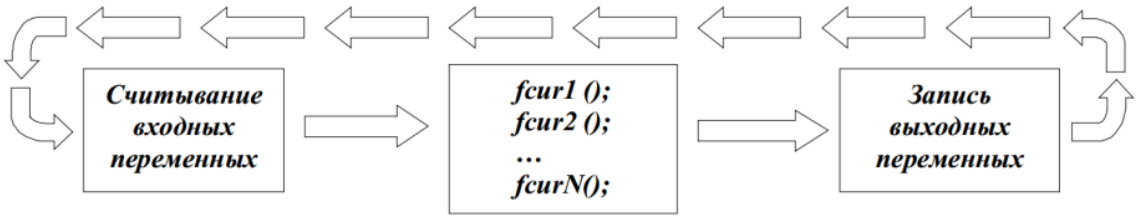

Рисунок 1.8 – Цикл гіперавтомата з множиною оброблювальних блоків

Врахуємо, що в цьому випадку гіперпроцес є деяким варіантом кооперативної багатозадачності. У такій кооперативній багатозадачності кожен процес є по суті окремим потоком управління, а сам гіперпроцес – структурою логічного паралелізму на рівні декількох інструкцій (функцій-станів). Функції-стани описуються в стилі мови С за допомогою стандартних безумовних і умовних операторів switch, if-else довільної вкладеності.

При візуальному формалізмі опису гіперавтомату у вигляді графу

переходів поряд з станами можуть використовуватися гіперстани що об'єднують кілька станів, що мають ідентичну реакцію на одну і ту ж подію. [13]. При цьому замість зображення таких переходів в деякий стан з усіх станів, які охоплюються гіперстаном, зображується тільки один перехід з гіперстану в зазначений стан (узагальнення переходів). Гіперстани теоретично можуть мати довільну глибину вкладення. Переходи з гіперстану пов'язані з усіма рівнями укладення в нього. Гіперстани можуть об'єднувати послідовні стани (OR-стани) і паралельні стани (AND-стани). У першому випадку, перейшовши в гіперстан, автомат може перебувати тільки в одному зі станів (або в одному стані, або в іншому), а в другому випадку в гіперстані «виникає необхідність в паралельних станах». зображуваних в «ортогональних (незалежних) регіонах».. На кожній дузі діаграми може бути зазначена подія, що викликає перехід, і логічна умова, «яка охороняє» цей перехід. На дузі також можуть бути вказані дії, одноразово виконуються на переході, і що формуються на переході події. У діаграмах можуть застосовуватися також псевдостани, наприклад, умовні (С), термінальні (Т) і історичні (Н), які не є реальними станами. Якщо гіперстан містить історичний псевдо-стан, то при переході в цей гіперстан «управління передається того стану, в якому система перебувала в даному гіперстани в останній раз» [13]. Використання узагальнення переходів і історичного псевдо-стану дозволяє, наприклад, ефективно уточняти переривання, пов'язане з переходом системи з будь-якого стану, що належить гіперстану, в якийсь інший стан, в якому виконується обробка переривання, і продовження роботи системи (після обробки переривання) з того стану, в якому настав переривання.

Таким чином можна визначити, що гіпераватомат – це автомат, який має у своєму складі гіперстани. Гіперстан – сукупність станів автомату, переходи між які здійснюються безумовно з урахуванням тільки часу існування кожного із станів. Стани автомата, які входять до гіперстану, повністю або частково зациклені. Вхід до гіперстану відбувається завжди до першого стану з множини станів гіперстану, незалежно від того, з якого зі станів основного

автомату здійснюється перехід до гіперстану. Вихід з гіперстану відбувається або по закінченні часу існування гіперстану, або при наявності «зовнішньої» події (які формується поза множиною вихідних оповіщувальних сигналів автомату), незалежно від того, в якому зі станів гіперстану знаходиться автомат у цей момент часу.

Гіперстан не має окремих вихідних змінних, використовуються тільки вихідні змінні основного автомату. Вихідні змінні основного автомату можуть використовуватися у гіперстані та навпаки. У гіперстані можуть змінюватися оповіщувальні сигнали основного автомату, але вихід з гіперстану здійснюється або за параметром часу, або за зовнішнім сигналом (подією), який не є оповіщувальним для будь-якого зі станів автомату. Зображення гіперстану на графі переходів автомата показано на рис.1.9

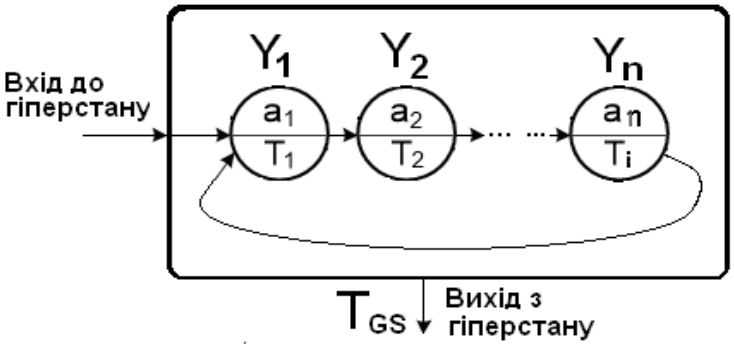

Рисунок 1.9 – Графічна модель гіперстану в графі переходів автомату

1.7 Модель пристрою керування освітлення пішохідних переходів

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.

Одним з поширених способів вхідного опису моделі спеціалізованого цифрового обчислювального пристрою обробки даних і логічного керування

є абстрактний кінцевий автомат, який представляється п'ятіркою $W =< X, A, Y, \delta, \lambda >$, 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. Найбільш вживаною формою представлення абстрактного автомату є таблиця переходів-виходів (ТПВ) або граф переходів автомату (state diagram) [7].

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.

Тривалість автоматного такту при цьому може бути одноковою для всіх стані автомату та залежати тільки від частоти синхропослідовності Clk, може бути різною для кожного стану та мати тривалість T.

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.

Для використання в подальшому викладенні поняття «темпорального

графу переходів» треба зробити уточнення деяких понять та визначень.

1. Стан автомату – математична абстракція, яка визначається набором стійких параметрів, який характеризується врахуванням передісторії динамічної системи та часом існування вказаного набору параметрів.

2. Вхідні сигнали структурного автомата поділяються на два види : дії та події Дії (effect) відповідають вхідним змінним автомату і існують, як правило, протягом всієї тривалості автоматного такту і можуть впливати на вихідні сигнали автомату (для автомата Мілі). Події (events) можуть мати тривалість менше автоматного такту і впливають на зміну стану автомата. Прикладом подій може бути синхропослідовність або зовнішній керуючий сигнал.

3. Тривалість кожного автоматного такту T_i приписується стану автомату (вершині графу переходів для автомату Мура). Таким чином, T_i визначає період часу, протягом якого стан автомату не змінюється. Події можуть перервати знаходження автомату в і-му стані та визначити перехід безпосередньо у момент появи події.

4. Вихідні сигнали автомату можуть впливати на події (умови скидання).

5. Гіперавтомат – це автомат, який має у своєму складі гіперстани. Гіперстан – сукупність станів автомату, переходи між які здійснюються безумовно з урахуванням тільки часу існування кожного із станів. Стани автомата, які входять до гіперстану, повністю або частково зациклені. Вхід до гіперстану відбувається завжди до першого стану з множини станів гіперстану, незалежно від того, з якого зі станів основного автомати здійснюється перехід до гіперстану. Вихід з гіперстану відбувається або по закінченні часу існування гіперстану, або при наявності «зовнішньої» події (які формується поза множиною вихідних оповіщувальних сигналів автомату), незалежно від того, в якому зі станів гіперстану знаходиться автомат у цей момент часу.

6. Гіперстан (його реалізація у програмному коді) є структурою,

інверсною до конструкції процесу (process) у мовах опису апаратури. У процесі обчислюються значення сигналів та змінних незалежно від параметру часу. У гіперстані напроти функції переходів обчислюються виключно від параметру часу. Стани автомату, які входять до гіперстану, відображуються моделлю Мура без вхідних сигналів $Y(t) = g(Z(t))$, $Z(t+1) = f(Z(t), T)$.

7. Гіперстан не має окремих вихідних змінних, використовуються тільки вихідні змінні основного автомату. Вихідні змінні основного автомату можуть використовуватися у гіперстані та навпаки.

8. У гіперстані можуть змінюватися оповіщувальні сигнали основного автомату, але вихід з гіперстану здійснюється або за параметром часу, або за зовнішнім сигналом (подією), який не є оповіщувальним для будь-якого зі станів автомату.

2 ПРИСТРОЇ КЕРУВАННЯ ОСВІТЛЕННЯМ ПІШОХІДНИХ ПЕРЕХОДІВ

2.1 Периферійне обладнання для освітлення пішохідних переходів

Зовнішнє освітлення міста є важливою та невід'ємною складовою інженерно-транспортної інфраструктури міста. Правильно спроектоване освітлення площ і вулиць – це комфорт і безпека людей в темний час доби. Про важливість зовнішнього освітлення говорить факт наявності в Державних будівельних нормах України окремого розділу, присвяченого питанням нормування зовнішнього освітлення.

Загальновідомо, що пішохідні переходи, навіть обладнані світлофорами, є зонами підвищеної небезпеки, яка збільшується у декілька разів в умовах поганої видимості – у сутінках або в нічний час доби. Обладнання пішохідного переходу спеціальним вуличним освітленням може істотно підвищити безпеку і знизити ризик для учасників руху.

При встановленні спеціального вуличного освітлення рекомендується враховувати ряд особливостей зорового сприйняття об'єктів в недостатньо освітлених місцях. Відомо, що світло, яке відрізняється по колірності від загального вуличного освітлення, має додаткову сигнальну дію. Колірну тональність освітлюваного простору визначає колірна температура джерела світла, тобто джерело світла в освітлювальному приладі на пішохідному переході повинно відрізнятися по колірній температурі від інших джерел світла, що освітлюють проїжджу частину, створюючи ділянку, що відрізняється по гаммі кольорів.

Відомо, що водії краще розрізняють пішоходів, коли ті з'являються у якості світлих об'єктів на темному фоні (тобто при позитивному контрасті). Цього можна досягти, якщо вуличний світильник розташувати між водієм і пішоходом, причому так, щоб його світло падало у напрямі руху автомобіля.

Залежно від типу світильників їх необхідно розташовувати на висоті від 50 до 100% стандартної опори.

В останніх законах та нормативно-правових актах України на проблеми зовнішнього освітлення вказано як в контексті розробки невідкладних заходів по зменшенню дорожньо-транспортних пригод (ДТП), так і в складі розв'язання питань благоустрою територій [1].

Освітлення наземних пішохідних переходів повинне забезпечувати пішоходам безпечний перетин проїжджої частини і можливість бачити перешкоди і дефекти дорожнього покриття. Для позначення зони переходу застосовують джерела світла з колірністю, контрастною по відношенню до колірності джерел світла основного освітлення вулиці.

На пішохідних переходах в одному рівні з проїжджою частиною вулиць і доріг категорій А і Б норма середньої освітленості h має бути в 1,5 разу вище, ніж на проїжджій частині, що перетинається. Підвищення рівня освітленості досягають зменшенням кроку опор, установкою додаткових або потужніших освітлювальних приладів (ОП).

Світлорозподілення ОП і їх орієнтація відносно наземного пішохідного переходу повинні забезпечувати контраст пішохода з фоном (проїжджою частиною) і не викликати засліплення водіїв. Розміщують ОП перед переходом по відношенню до транспорту, що наближається, направляючи світло на пішохода з боку водія. На дорогах з двостороннім рухом ОП встановлюють перед перехрестям відносно обох напрямів руху. Для зниження засліплювання водіїв рекомендується використовувати ОП з асиметричним світлорозподіленням (кососвітлові прилади).

Для попередження ДТП з тяжкими наслідками мають значення дотримання наступних десяти основних правил безпеки:

– вночі на пішохідному переході повинно бути видно, як вдень, учасники руху повинні бачити один одного;

– пішохідний перехід повинен освітлюватися в 1,5 разу краще за вулицю;

– пішоходи повинні контрастувати зі світловим і колірним фоном;

– світло повинне йти з боку водія;

– засліплення учасників руху має бути виключене;

– водій повинен заздалегідь бачити далекий пішохідний перехід;

– пішохідні переходи мають бути позначені знаками;

– світлофори, регулюючі рух - основа безпеки;

– має бути стоп - лінія, лежачий поліцейський;

– огорожі повинні обмежити можливість пішоходів переходити дорогу поза пішохідним переходом;

Правила перетину пішоходами проїзної частини регулюються правилами дорожнього руху, які визначають наступне:

– у місцях, де рух регулюється, пішоходи повинні керуватися сигналами регулювальника або пішохідного світлофора, а при його відсутності - транспортного світлофора;

– на регульованих пішохідних переходах пішохід зобов'язаний підкорятися сигналам світлофора або регулювальника.

Регульовані пішохідні переходи оснащуються пішохідними світлофорами, в секціях яких є символічне зображення людини. Такі світлофори мають дві секції - червону і зелену. Допускається застосування звукової сигналізації, яка представляє собою серію однотонних звукових сигналів. При цьому інтервали між звуковими сигналами можуть скорочуватися номері того, як закінчується час дії сигналу світлофора.

Система автоматичної індикації пішохідного переходу призначена для освітлення і посилення візуального сприйняття водіями нерегульованих пішохідних переходів в темний час доби. Система складається з наступних складових частин:

– дорожнього знаку "пішохідний перехід" з підсвічуванням полязнаку;

– детектору знаходження пішохода в зоні підготовки переходу;

– комплекту синхронізації включення знаків і освітлення;

– світлодіодних вуличних світильників із стійками освітлення;

– кнопки виклику пішохода.

У зоні нерегульованого пішохідного переходу встановлюються опори освітлення заввишки не менше 5 метрів з обох боків проїжджої частини, на яких монтуються знаки "пішохідний перехід", детектори пішоходів або кнопки виклику пішоходів, світлодіодні світильники із спрямованою діаграмою освітленості на перехід, і, при необхідності, елементи системи автономного живлення (сонячні панелі і акумулятори). Система, досягнувши низьких рівнів освітлення, працює таким чином. Пішохід, готовий до переходу або самостійно через кнопку виклику, або автоматично детекторами пішоходів активує систему. При цьому включається миготливе підсвічування знаку пішохідного переходу і світлодіодні світильники, освітлюючи зону пішохідного переходу на проїжджій частині. Включення відбувається синхронно на обох стійках, триває обмежений час, необхідний для перетину дороги пішоходом, і потім підсвічування і освітлення вимикаються.

На рис.2.1 наведені приклади конструкцій освітлення пішохідних переходів.

Рисунок 2.1 – Приклади конструкцій освітлення пішохідних переходів

Поряд з пішохідними світлофорами допускається встановлення пішохідного викличного табло (ПВТ) – пристрою для виклику пішоходами сигналу світлофора, який дозволяє рух та освітлення пішохідного переходу.

Пішохідне викличне табло – інтелектуальний технічний виріб, що формує сигнал виклику, який дозволяє пішоходам безпечно пересуватися через проїжджу зону. ПВТ зручне для усіх учасників транспортного процесу, оскільки зелене світло спалахує виключно на вимогу того, що пішоходи переходять дорогу. ПВТ призначене управління викличною фазою дорожнього контролера на світлофорних об'єктах пішохідного переходу і світловою індикацією очікування, тобто для включення зеленого сигналу пішохідного світлофора по запиту пішохода. ПВТ виконує функцію формування сигналу запиту виклику дозвільного сигналу пішоходами. Формування заявки на виклик пішохідного сигналу здійснюється при натисненні будь-якої з кнопок. При цьому на дорожній контролер надходить сигнал "Виклик". У ПВТ передбачений індикатор, включення якого вказує на те, що запит прийнятий контролером. Індикатор горить до моменту включення дозвільного сигналу на пішохідному переході.

2.2 Технічне завдання на проектування

Метою розробки є проектування мікроконтролерного пристрою керування освітленням пішохідного переходу, який окрім включення зовнішнього освітлення переходу забезпечує підсвічення смуг «зебри» переходу, що забезпечує супроводження пішохода при перетині проїжджої частини автодороги. Пристрій повинен відповідати всім критеріям систем логічного управління та системам реального часу.

Алгоритм роботи пристрою керування освітленням пішохідного переходу наступний. Пристрій включається вимикачем тільки у темний час доби. Безпосереднє включення освітлення здійснюється за командою пішохода шляхом натиснення кнопки пішохідного викличного табло. Після цього загоряється зовнішнє освітлення переходу та червоне світло дорожнього світлофору для автотранспорту. Дається час на підготовку пішохода до переходу, а транспорт за цей час має залишити «зебру» переходу. Після цього

починається підсвічення смуг «зебри» переходу, яке супроводжує пішохода при перетині проїжджої частини автодороги. Для зручності перетину дороги з двох боків здійснюється зустрічне підсвічення пар смуг переходу до середини автодороги та назад. Швидкість підсвічення розраховується такою, що дозволяє пішоходу вільно здійснювати перетин автодороги та регулюється в залежності від кількості смуг руху автотранспорту. Після цього кнопка ПВТ блокується на час, достатній для забезпечення вільного руху транспортного потоку. Натиснення кнопки ПВТ під час блокування запам'ятовується, а подія включення світла здійснюється після закінчення часу блокування. Якщо за час блокування кнопка ПВТ не натискувалася, пристрій переходить до режиму чекання.

Фізично освітлені смуги переходу можуть бути реалізовані або за рахунок підвішених світильників, які імітують смуги переходу, або за рахунок настилу смуг переходу, що підсвічуються. Існуючі приклади реалізації таких пристроїв наведені на рис.2.2.

а) б)

Рисунок 2.2 – Технічна реалізація підсвічення смуг пішохідного переходу:
а) підвішені освітлені смуги переходу – розробка Артемія Лєбєдєва;
б) накладні смуги переходу – розробка фірми IBM

Пристрій має бути простий та дешевий і використовувати розповсюджений тип мікроконтролера. Макет пристрою має бути реалізований на платі відлагодження з використанням кольорових

світлодіодів. Програмна реалізація пристрою повинна бути витримана за принципом автоматного програмування та побудови автоматного шаблону з урахуванням всіх особливостей мікроконтролерної бази (подійна система/система переривань та механізми обробки подій/переривань) та урахуванням того, що дана система є системою реального часу та автоматний шаблон має опис часового автомату.

2.3 Принципи автоматного програмування

Як уже згадувалося, сучасні системи логічного управління складаються з двох складових: апаратної і програмної.

Особливість апаратури полягає в тому, що вона розробляється одними фахівцями, а виготовляється (комплексується) – іншими. Це призводить до необхідності проводити розробку у формі проектування, пов'язаного з випуском великого числа різнотипних схем і інших конструкторських документів, відповідних діючим стандартам і що досконально відбивають усі аспекти життєвого циклу апаратури. Тому навіть через багато років ця частина систем при необхідності може бути порівняно легко модифікована, у тому числі і іншими фахівцями.

Принципово інша ситуація має місце при створенні програмного забезпечення, оскільки воно розробляється і "виготовляється" одними і тими ж фахівцями. Тому зазвичай розробка програм не виконується у формі проекту тієї ж міри подробиці, як це робиться для апаратури, що часто призводить до значних труднощів при необхідності їх модифікації.

Для систем логічного управління при введенні вхідних параметрів по запиту, запропонована SWITCH-технологія, призначена для алгоритмізації і програмування завдань логічного управління [2]. При цьому відповідний підхід до програмування був названий «автоматним програмуванням». Одна з центральних ідей автоматного програмування полягає у відділенні опису логіки поведінки (за яких умов необхідно виконати ті або інші дії) від опису

його семантики (власне сенсу кожного з дій). Крім того, опис логіки при автоматному підході жорстко структурований. Ці властивості роблять автоматний опис алгоритмів логічного управління наочним і ясним.

Основні принципи автоматного програмування викладені нижче.

1. Представлення автоматних моделей.

Як базове використовується поняття "автомат", а не "клас", "об'єкт", "алгоритм" або "агент" і застосовується модель змішаного автомата, для опису поведінки якого використовується відповідний граф переходів. У загальному випадку автомати розглядаються не ізольовано, а як складові частини взаємозв'язаної системи – системи взаємозв'язаних автоматів, поведінка якої формалізується за допомогою системи взаємозв'язаних графів переходів.

Графи переходів (часові або темпоральні графи переходів) автомата (temporal state diagram) є не тільки візуальним відображенням алгоритму функціонування автомату, але й повною його математичною моделлю.

Стани кожного автомата спочатку визначаються по виділених технічних станах об'єкту управління або його частини, а при великій їх кількості – по алгоритму управління, побудованому в іншій нотації (наприклад, у вигляді граф-схеми алгоритму). На етапі проектування (на відміну від традиційного програмування) використовується кодування станів автомата.

Вхідні дії розділяються на події, які діють короткочасно (наприклад, натискання кнопок, сигнали датчиків), і вхідні змінні, що вводяться шляхом опитування. Вхідні дії доцільно реалізовувати у вигляді вхідних змінних, а події застосовувати для скорочення часу реакції системи. При цьому одна і та ж вхідна дія може бути одночасно представлена і подією і вхідною змінною, Усі вихідні дії є діями, а не діяльностями. Групи вхідних і вихідних дій зв'язуються із станами, виділеними для кожного автомата.

Переривання обробляються операційною системою і передаються програмі у вигляді повідомлень, а після цього обробляються як події за допомогою відповідних обробників. Деякі вхідні змінні можуть формуватися в результаті порівняння вхідних аналогових сигналів з еталонами.

2. Реалізація функціонування автоматів.

Автомати можуть запускатися одноразово з передачею якої-небудь події або багаторазово (у циклі) з передачею однієї і тієї ж події. При реалізації системи необхідно враховувати, що функції, які реалізовують автомати, не реєнтерабельні (не допускають повторного запуску до їх завершення). Кожен автомат при запуску реалізує не більш за одного переходу. Після обробки чергової події автомат зберігає свій стан і "засинає" до появи наступної події, якщо таймаут дорівнює ∞.

Дуги і петлі графів переходів позначаються довільними логічними формулами, які можуть містити вхідні змінні і предикати, перевіряючи номери станів інших автоматів і номера подій. Дуги і петлі окрім умов переходів можуть містити список послідовно виконуваних вихідних дій.

Вершина графа переходів може містити список вкладених автоматів, що послідовно запускаються, і список послідовно виконуваних вихідних дій.

3. Програмна реалізація.

На етапі реалізації будується програма (на відповідній мові програмування або опису апаратури), в якій графи переходів, вхідні змінні, обробники подій і вихідні дії реалізуються у вигляді функцій. Крім того, програма містить допоміжні модулі, що складаються з наборів відповідних функцій (наприклад, модуль управління таймерами). Обробники подій містять виклики підпрограм, що реалізовують графи переходів (автомати), з передачею їм відповідних подій. Функції вхідних і вихідних дій викликаються з підпрограм, що реалізовують автомати. Функції, що утворюють допоміжні модулі, викликаються з функцій вхідних і вихідних дій.

Кожен граф переходів формально і ізоморфно реалізується окремою функцією (підпрограмою), що створюється за шаблоном, який містить дві конструкції switch і оператор if. Перша конструкція switch викликає вкладені автомати і реалізує переходи і дії на дугах і петлях. Оператор if перевіряє, чи змінився стан, і якщо він змінився, друга конструкція switch активізує вкладені автомати і реалізує дії в новій вершині.

4. Взаємодія з оточенням.

Система взаємозв'язаних автоматів утворює системонезалежну (наприклад, від операційної системи) частину програми, яка реалізує алгоритм функціонування системи управління.

Реалізація вхідних змінних, обробників подій, вихідних дій, допоміжних модулів і призначених для користувача інтерфейсів утворює системозалежну частину програми;

5. Верифікація.

Кожен граф переходів перевіряється на досяжність, несуперечність, повноту і відсутність генеруючих контурів. Після реалізації графа переходів текст підпрограми повинен коригуватися для забезпечення безповторності опитування вхідних змінних, що позначають дуги, які виходять з одного стану. Таким чином, вирішується проблема "ризику".

6. Документування.

Детальне документування проекту по створенню програмного забезпечення дозволяє при необхідності вносити зміни до нього через тривалий термін після його випуску, у тому числі фахівцями, які не брали участь в проектуванні.

Склад документації; структурна схема системи; схема розробленого програмного забезпечення; роздруківки екранів інтерфейсів, призначених для користувача; переліки подій, вхідних змінних і вихідних дій; діаграма взаємодії автоматів; опис термінології і форми представлення, використовуваної в графах переходів; шаблон для реалізації графів переходів змішаних автоматів довільного рівня вкладеності; для кожного автомата: словесний опис (фрагмент технічного завдання), що розглядається як коментар, схема зв'язків автомата, граф переходів і початковий текст функції, що реалізовує автомат; алгоритми, наприклад у вигляді графів переходів, і початкові тексти допоміжних модулів і функцій, що реалізовують вхідні змінні, обробники подій і вихідні дії; протоколи для сертифікації програми, що виконують роль контрольних прикладів; керівництво програміста.

3 АВТОМАТНИЙ ШАБЛОН В РЕАЛІЗАЦІЇ СИСТЕМИ ЛОГІЧНОГО УПРАВЛІННЯ

3.1 Апаратні та програмні компоненти реалізації пристрою керування.

Для реалізації пристрою, що розробляється необхідно узгодити всі необхідні компоненти апаратної та програмної реалізації.

В якості апаратної реалізації використовується мікроконтролер F429ZIT6 сімейства STM32[14], який розведено на платі для розробки DISCO.

В якості програмної реалізації використовується САПР CubeMX для швидкої побудови каркасу у вигляді коду для опису системи з усіма необхідними системними налаштуваннями.

STM32CubeMX є візуальним графічним редактором для конфігурації мікроконтролерів сімейства STM32, що дозволяє генерувати код на основі мови Сі, використовуючи для цього графічних помічників[15].

Генерація коду CubeMX багата на вибір інтеграційних серед розробки (Integrated Development Environment (IDE)). Однією з найпопулярніших та більш сучасних серед є Keil for ARM (MDKARM5)[16].

Основною бібліотекою опису пристрою являється бібліотека слою апаратних абстракцій (Hardware Abstraction Layer (HAL)).

Це бібліотека для створення проектів на STM32, розроблена компанією STMicroelectronics у 2014 році. HAL дозволяє абстрагуватися від роботи з регістрами. Грубо кажучи, HAL – це обгортка над низькорівневими операціями. Звичайно ж це не скасовує необхідності розуміння пристрою мікроконтролерів, але значно знижує рівень входження [17].

Додатковим пристроєм, який не відноситься до реалізації керуючого пристрою, але дозволяє продемонструвати роботу пристрою у вигляді відображення сигналів (генерація часової діаграми (waveform))

використовується апаратний пристрій моніторингу сигналів – логічний аналізатор Saleae 8 Logic.

Логічний аналізатор – електронний прилад, який може записувати і відображати послідовності цифрових сигналів. Він використовується для тестування і налагодження цифрових електронних схем, наприклад, при проектуванні компонентів комп'ютерів і керуючих електронних пристроїв. На відміну від осцилографів, логічні аналізатори мають значно більше входів (зазвичай у проектах використовуються логічні аналізатори з 16 каналами та до кількох сотень), але при цьому такі електронні пристрої здатні показувати лише два рівня сигналу («0») і («1»), до яких іноді додано стан «Z» [18].

Для відображення результатів відлагодження пристрою за допомогою логічного аналізатора використовується відповідне ПО виробника – Saleae Logic 1.2.18 (актуальна версія програми). Вид програми зображено на рис.3.16-3.19.

3.2 Налаштування периферії МК

Логіка опису автоматного шаблону на базі мікроконтролера майже ідентична опису HDL-моделі. Існує невелика кількість відмінностей.

1. Інша мова програмування (C/C++).
2. Використання вже реалізованої внутрішньої периферії (порти вводу/виведення, таймери, АЦП і т.д.).
3. Використання принципів моделі подій/переривань та використання обробників переривань.
4. Додаткові внутрішні сигнали (змінні) для взаємозв’язку з обробниками переривань. Додаткові змінні зберігають побудову шаблону за логікою розділення на функцію переходів-виходів та функцію затримок.

З урахуванням цих відмінностей варто зауважити те, що реалізація системи тактування на МК лежить на вбудованому тактовому генераторі, частота якого складає 8 МГц. Без підключення внутрішньої периферії основна

функція тактового генератора – SysTick (рис.3.1). Це функція, яка підтримує роботу МК та периферії у цілому. Будь-які зміни та переналаштування SysTick можуть призвести до неконтрольованої роботи МК, що обов'язково призведе до несподіваних наслідків (hard fault, вихід з ладу МК та периферії, перехід МК у стан захисту).

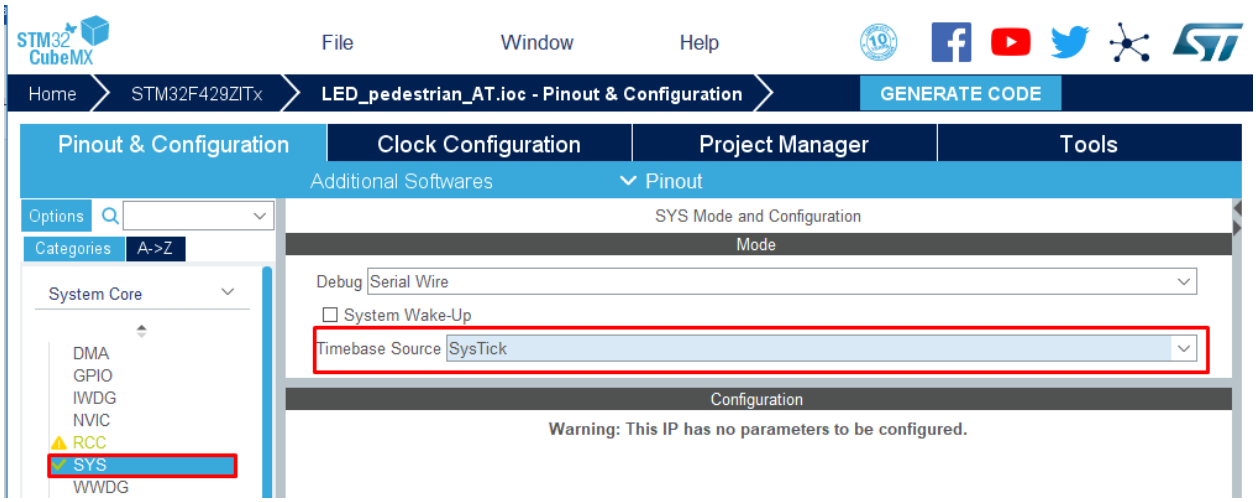

Рисунок 3.1 – Вибір SysTick в якості основної функції тактування МК

Правильним та грамотним рішенням цієї проблеми є використання таймеру МК для коректної реалізації тактування та функції затримок (рис.3.2).

Налаштування періоду відбувається за замовчуванням та надалі може змінюватись розробником в різних частинах опису пристрою.

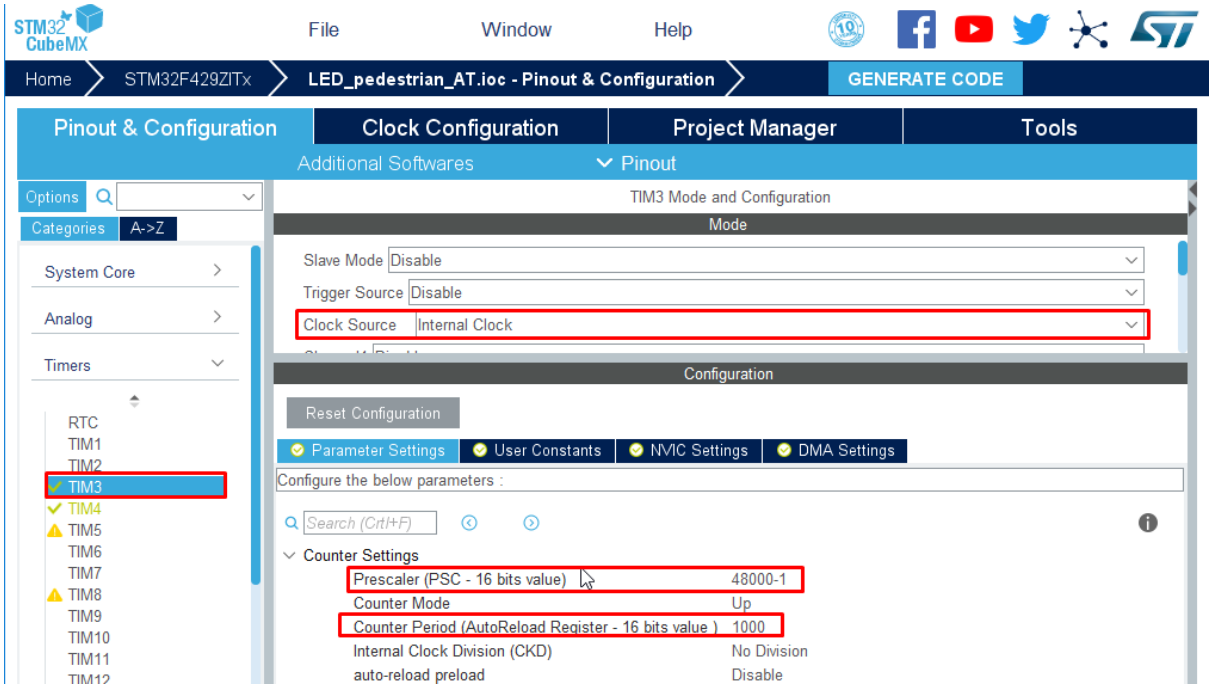

Рисунок 3.2 – Включення та налаштування таймера (TIM3) за замовчуванням

Налаштування таймеру та периферії (портів вводу/виведення) відбувається згідно з документацією на МК. Максимальна частота на таймері не повинна перевищувати 90 МГц, частота периферії – 45 МГц. Раціональним рішенням буде налаштування шини даних на таку тактову частоту, яка має націло ділитись на тактування від внутрішнього тактового генератора (рис. 3.4).

Перед налаштуванням тактування периферії необхідно вибрати високошвидкісний вбудований тактовий генератор. Такий генератор має генерацію сигналів, частота яких більше 1МГц (рис.3.3).

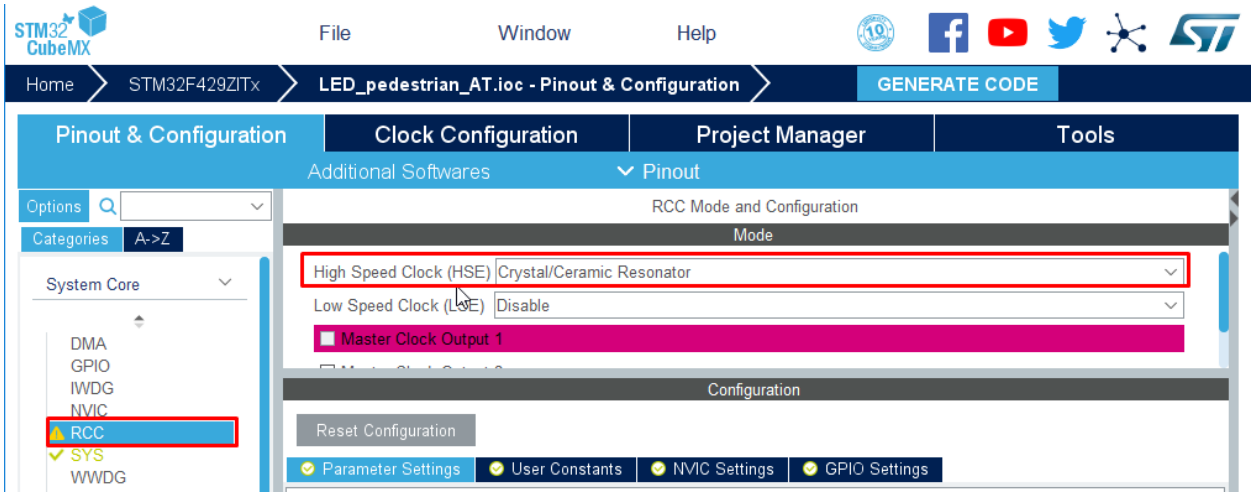

Рисунок 3.3 – Включення та налаштування внутрішнього тактування

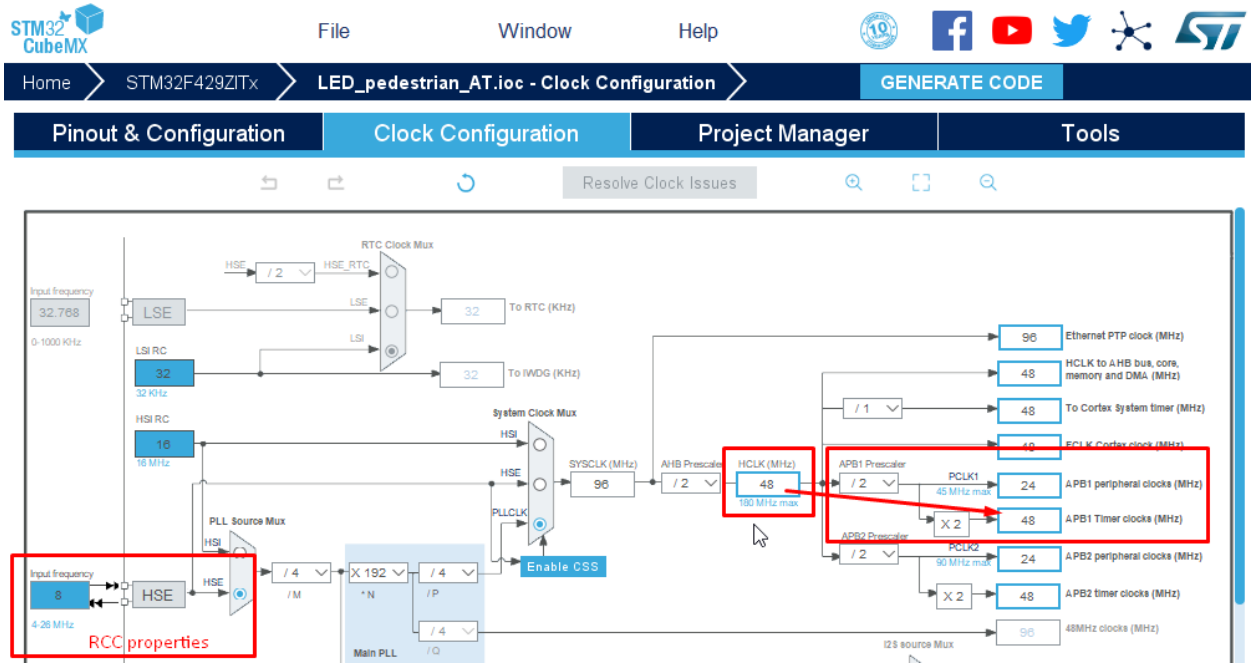

Рисунок 3.4 – Налаштування тактування периферії МК

Mux PLL-блок, зображений на рис.3.4 є системою фазового автопідстроювання частоти (ФАПЧ). В цьому блоці відбувається налаштування частоти з керованим коефіцієнтом множення (посилення або послаблення частоти сигналу в блоках МК, де це необхідно) для подальшої стабілізації та подачі необхідної частоти до периферії. Блок автоматично налаштовується системою.

Для вводу або виведення даних в або з МК використовуються дискретні порти вводу/виведення (GPIO). Напрямлення інформації налаштовується розробником (рис.3.5).

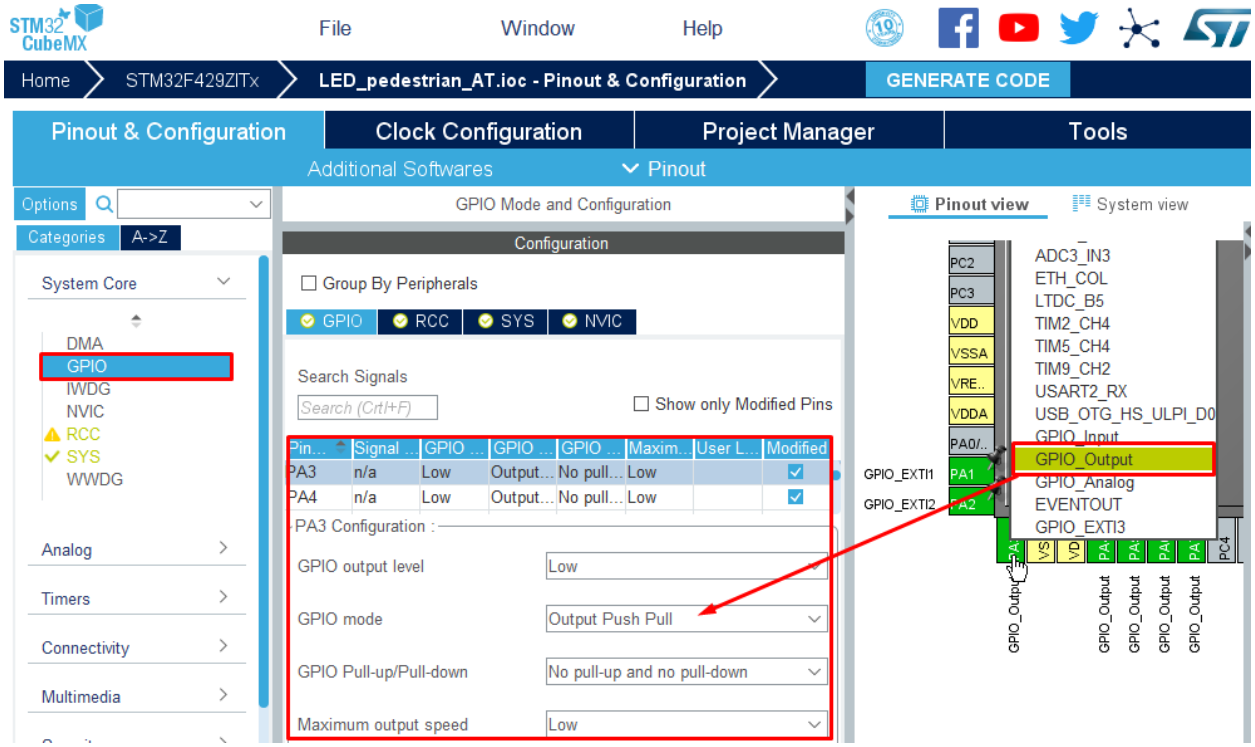

Рисунок 3.5 – Налаштування портів на виведення інформації

Кожний пін портів вводу/виведення має змогу бути включеним до системи подій/переривань. Сигнал про зовнішню подію та переривання налаштовуються розробником аналогічно налаштуванню GPIO на вхід або вихід (рис.3.6).

Реагування МК на зовнішню подію та переривання, за замовчуванням, відбувається за позитивним фронтом. Цей параметр можна змінити як на задній фронт, так і на змогу реакції на передній та на задній фронт одного й того ж сигналу.

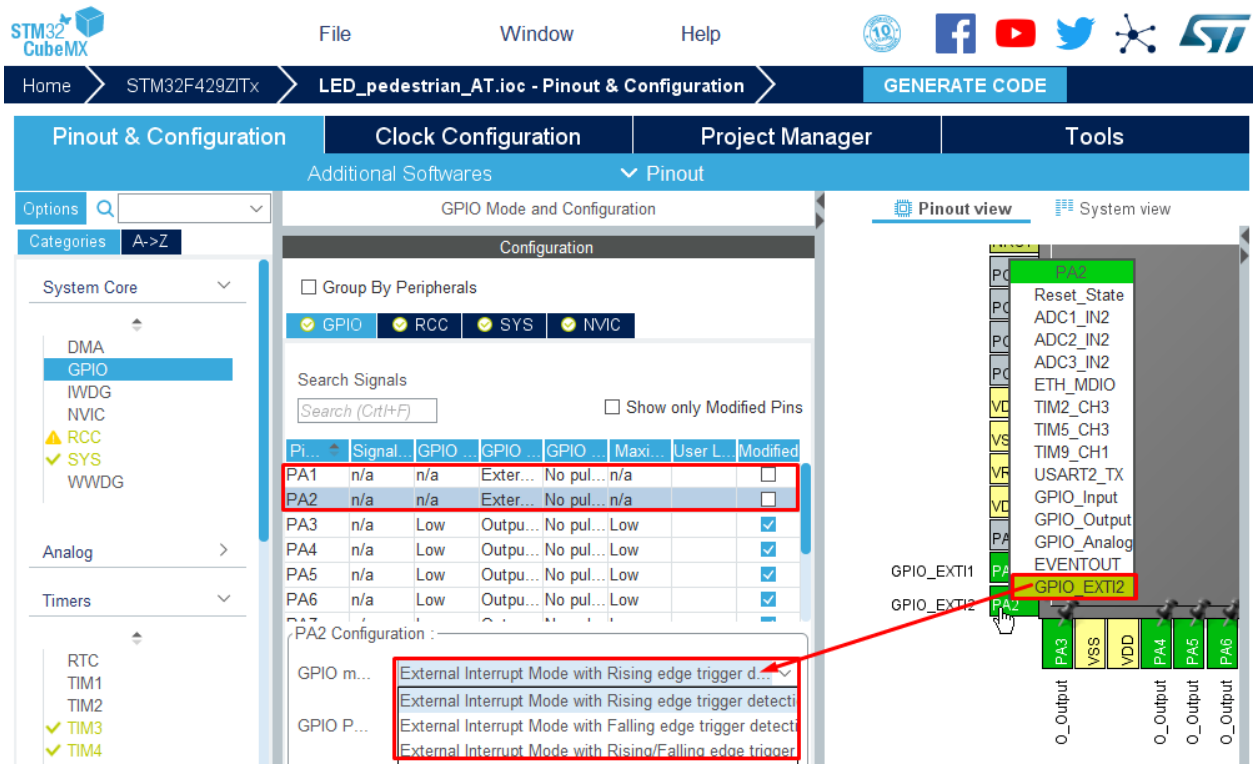

Рисунок 3.6 – Налаштування портів на реакцію щодо зовнішніх подій та переривань

Одного налаштування портів на реакцію щодо подій та переривань недостатньо для реагування на МК на них. Для цього необхідно дати дозвіл системі. Для цього і існує вбудований контролер переривань (NVIC (рис.3.7)), який містить в собі таблицю векторів переривань (IVT), яка на кожну подію та переривання (IRQ) має свій власний обробник (ISR), що і є ініціатором дій зі сторони МК.

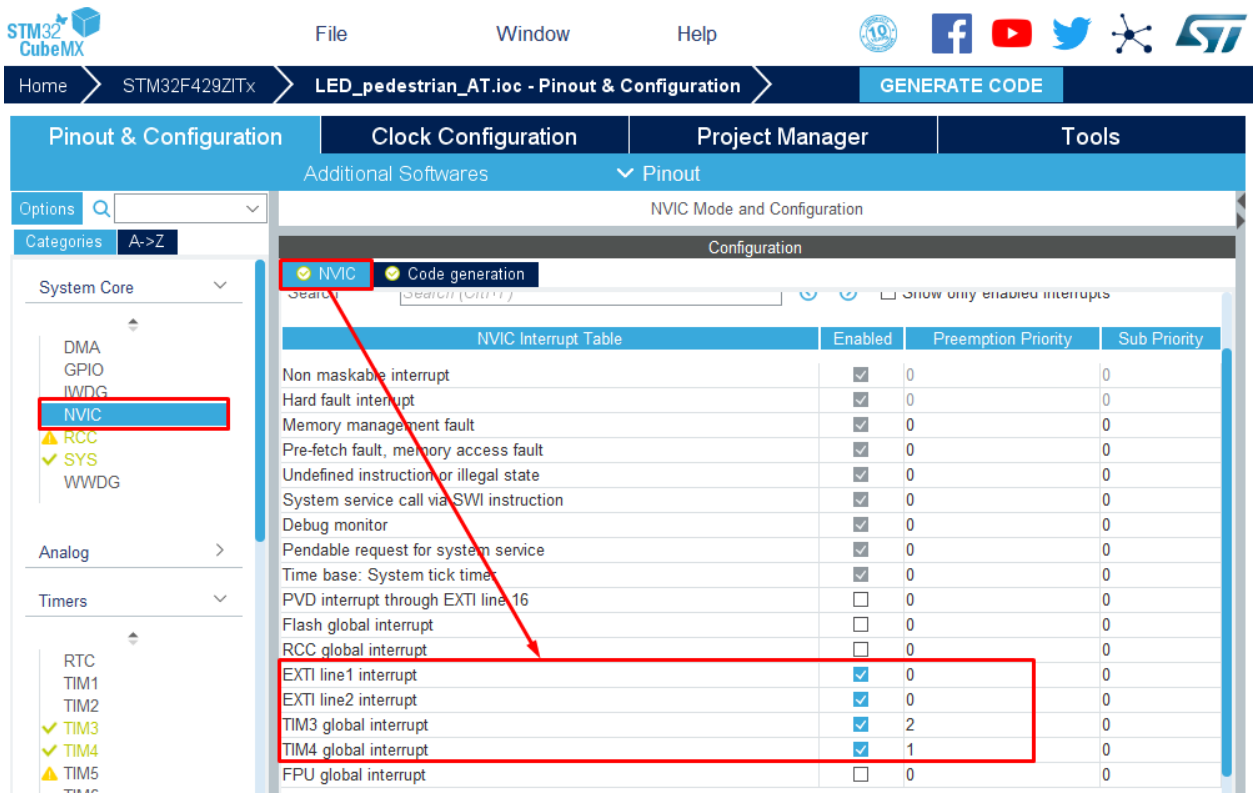

Рисунок 3.7 – Створення обробників переривань з налаштуванням їх пріоритету

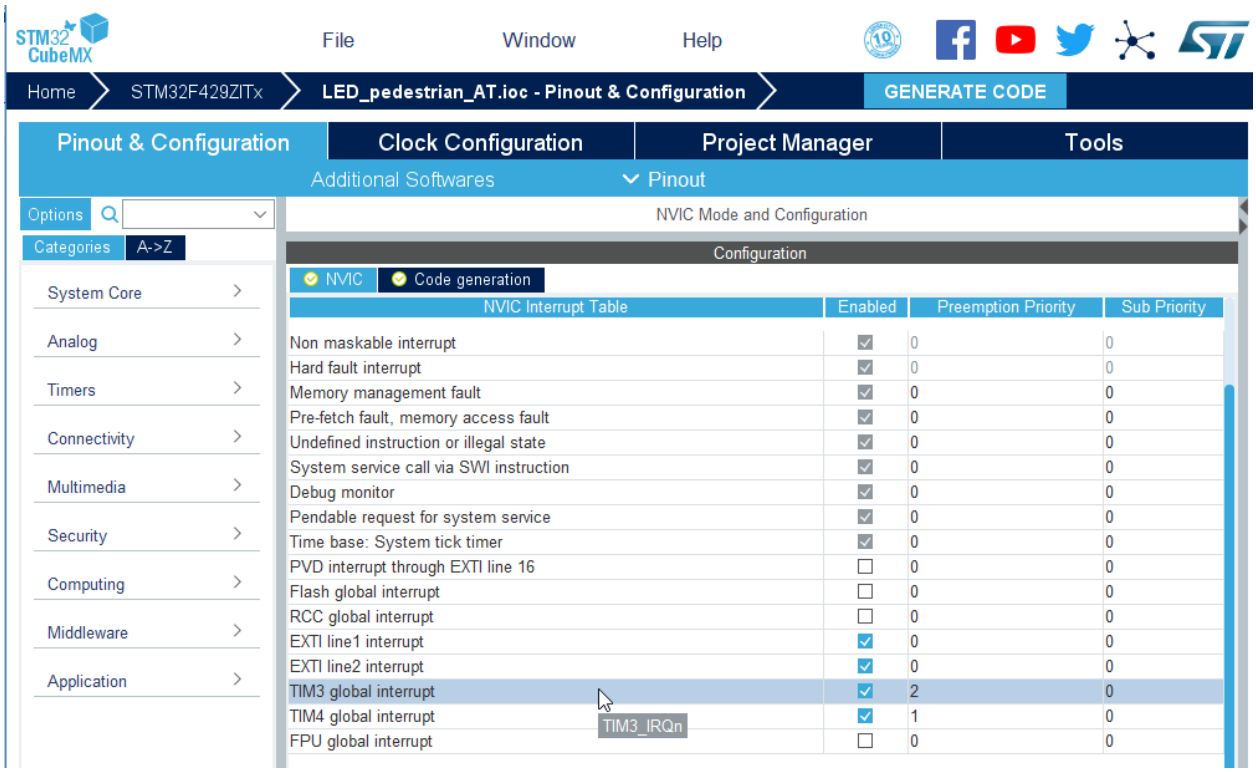

Рисунок 3.8 – Пріоритет переривань

На рис.3.8 відображено розстановка пріоритетів всім перериванням у МК. Пріоритет переривань досить важливий для систем, які повинні відпрацювати певний проміжок часу без можливості реагування на втручання інших подій. За правилом зовнішні події завжди пріоритетніші за події, які виникають у внутрішній периферії МК, підключеній розробником. Пріоритет системних переривань, власних переривань NVIC зарезервовано системою і завжди найвищий (0). Розробник має право на встановлення найвищого пріоритету власним перериванням без загрози збоїв обробки зарезервованих переривань. Також розподіл переривань за пріоритетом дає змогу коректно їх ініціювати при створенні проекту для опису пристрою.

3.3 Опис стандартного автоматного шаблону для пристрою на базі мікроконтролеру

Програмну реалізацію пристрою, який розробляється, відтворено на основі автоматного програмування з використанням автоматного шаблону для систем реального часу. Принцип дії пристрою можна відтворити за допомогою графу переходу, зображеного на рис.3.9.

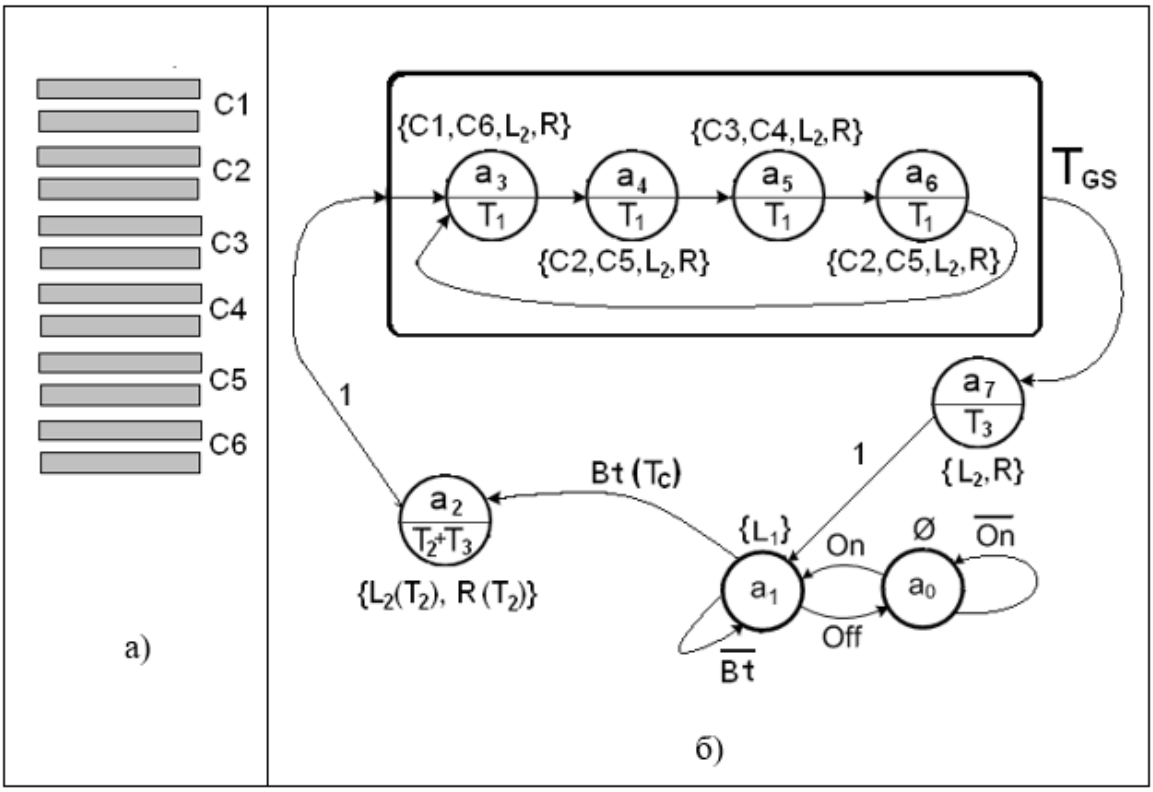

Рисунок 3.9 – Модель керуючого автомату освітлення пішохідного переходу за стандартним описом АШ на МК: а) вихідні сигнали підсвічення смуг; б) граф переходів

Основний режим роботи системи освітлення починається з натиснення пішоходом кнопки пішохідного викличного табло (ПВТ), яка ініціює подію Bt. При цьому слід враховувати, що час натиснення не може бути менше, ніж T_b (наприклад T_b = 0,1с). Після цього пристрій керування чекає час T_2, що дає змогу пішоходу підготуватися до переходу (T_2= 5с). Після цього включається основне освітлення переходу на стовпі (сигнал L2 з затримкою T_2), загоряється червоне світло дорожнього світлофора (сигнал R з затримкою T_2) і надається час T_3 для того, щоб транспорт зупинився перед переходом. Таким чином таймаут для стану а2 є (T_2+T_3), тобто (T_{a2} = 5с+5с). Після цього починається зустрічне підсвічення пар смуг переходу з протилежних сторін дороги до середини та назад. Кожні 2 смуги підсвічуються на час T_1 (T_1 = 3с). Для дороги на дві смуги (12 смуг «зебри») це складає 15 секунд, за рахунок повернення до

початкового стану { C1-C6, C2-C5, C3-C4, C2-C5, C1-C6 }. Перехід закінчено. Після цього знов надається час T_3, щоб пішоходи залишили проїжджу частину дороги, було відключене верхнє освітлення переходу та червоне світло дорожнього світлофора. Після цього кнопка Bt (у програмній реалізації має сигнатуру BTN) блокується (забороняється реакція на подію Bt) на час до закінчення вхідних обмежень Tc (Tc =[1, 30]c) для того, щоб дати можливість рухатись транспорту і не заблокувати дорожній рух шляхом безперервного натиснення кнопки Bt. Натиснення кнопки ПВТ під час блокування запам'ятовується, а подія включення світла здійснюється після закінчення часу блокування. Після цього пристрій готовий до нового натиснення кнопки.

Таким чином загальний цикл роботи пристрою керування освітленням складає $T = T_2 + T_3 + 5*T_1 + T_3 + T_c = 5+5+3*5+5+30 = 60$c.

Варто зауважити деякі аспекти коректної та грамотної реалізації шаблону.

Лістинг 3.1 – Опис визначення станів та затримок

```
/*States definition with numbers coding (0-7)*/
typedef enum {a0,a1,a2,a3,a4,a5,a6,a7} States;

/*All delays in ms (0-6 + DELAYS_COUNT that shows the count of delays)*/
typedef enum {T0,T1,T2,T3,Tb,Tc,DELAYS_COUNT}Delays;
```

Для компактності та читабельності коду, для відсутності необхідності зміни налаштувань портів вводу/виведення власноруч існує поняття макрос, який сприяє спрощенню сприйняття коду розробником. Він створює псевдонім, який використовується для опису різних даних. Такий підхід повністю використовується у генерації проекту в тих місцях, де розробнику, наприклад, не потрібно знати код натиснутої кнопки, а потрібно знати яка саме кнопка була натиснута. Також за допомогою макросу можна замінити виклик певної функції.

У програмній реалізації макроси використані для іменування вихідних сигналів, виклику функції, яка перевіряє чи натиснута кнопка та дозволу або заборони переривання.

Лістинг 3.2 – Опис визначення входів та виходів, дозвіл та заборона переривання від натискання на кнопку BTN

```
/*All the LEDs that locates in LED control block*/
#define BTN_LIGHT       GPIO_PIN_3
#define RED_LIGHT      GPIO_PIN_4
#define LIGHTER         GPIO_PIN_5
#define C1              GPIO_PIN_6
#define C2              GPIO_PIN_7
#define C3              GPIO_PIN_8
#define C4              GPIO_PIN_9
#define C5              GPIO_PIN_10
#define C6              GPIO_PIN_11

/*Allow to process the BTN pushing*/
#define BTN_READY            HAL_NVIC_EnableIRQ(EXTI2_IRQn);

/*Forbid the BTN pushing processing*/
#define BTN_NOT_READY HAL_NVIC_DisableIRQ(EXTI2_IRQn);

/*CHECK if the system is ON/OFF*/
#define BUTTON_ON HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);

/*CHECK if the btn is pushed*/
#define BTN HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2);
```

У головному файлі програмної реалізації пристрою (main.c) відбувається всі основні процеси шаблонізації. По-перше, завдяки створення переліку всіх затримок у заголовному файлі, у головному файлі створюється масив, який зберігає значення цих затримок в порядку переліку.

Створюються та ініціюються всі необхідні змінні для реалізації автоматного шаблону за всіма відмінностями (змінна для підрахунку часу перебування у гіперпроцесі, змінні визначення поточного та наступного стану та змінні для зв’язку переривань від кнопки з функцією переходів-виходів).

Лістинг 3.3 – Визначення необхідних змінних

```
//Array with a delay values
//
uint16_t delays[DELAYS_COUNT] = {1000,3000,5000,5000,101,30000};
/*
      Special variable for delay function to execute the transition
      a5-a4-a3-a6
      volatile specificator tells the program that
      the variable value can be changed in another part of main program
      (e.g.interrupts)
      but the main cycle will know it
*/
volatile uint16_t walkTime;

volatile States state = a0;   //Current state
States nextState;             //Next state

//A special checkers for each pushed/unpushed button in system
//
volatile uint8_t isButtonOnPushed;
volatile uint8_t isBtnPushed;
```

Після визначення та ініціації всіх необхідних змінних та даних, наступним кроком є опис функції переходів-виходів. Функція аналогічна процесу в HDL-моделі, але особливість у тому, що при реалізації даної функції на МК, створюється тісний зв’язок як із перериваннями від зовнішніх подій (натискання кнопки BUTTON_ON та BTN), так і з перериванням від таймеру (функція затримок).

У програмній реалізації даного пристрою переривання для увімкнення системи має аналогічність з перериванням від кнопки початку переходу. По-перше дуже гарним тоном є програмна реалізація системи проти брязкоту контактів, яка зменшує шанси неконтрольованого замикання та появи нестабільного позитивного фронту на вході. По-друге варто зауважити, що використання переривання для увімкнення системи (стан a0) є частою практикою в реалізації систем. Більше всього таким способом реалізують вихід МК з режиму сну, якщо всі умови реалізації орієнтуються на економію електроенергії навіть при малих показниках споживання струму (за документацією споживання струму МК становить 40 мА при максимальній частоті роботи у 180 МГц).

Також варто зауважити, що для спрощення дозволу натискання кнопки, створено її підсвічення (BTN_LIGHT).

Лістинг 3.4 – Опис станів a0 та a1 у функції переходів

```
void StateTransition()
{
	//Transitions function
	//
	switch(state)
	{
		case a0:		/*Block with transition description*/

			if(isButtonOnPushed == 0)	/*BUTTON_ON push/unpush
								checker*/
			{
				nextState = a0;
			}

			else
			{
				nextState = a1;
			}

		break;

		case a1:

			if(isButtonOnPushed == 0)
			{
				nextState = a0;
			}

			else
			{
				if(isBtnPushed == 0) /*BTN push/unpush checker*/
				{
					nextState = a1;
				}

				else
				{
					nextState = a2;
				}
			}
		break;
	…

}
```

Тісний зв'язок функції переходів з перериваннями пов'язан з додатковим змінними, які після перевірки сигналів на контрольоване натискання певний проміжок часу отримують позитивний результат натискання кнопок та зберігають його до того моменту, коли значення змінних на даний час роботи пристрою не важливі. Оскільки функція переходів-виходів виконує свою роботу безперервно – не варто перевіряти в ній натискання кнопки. Це пов'язано з тим, що сигнал від натискання кнопки повинен бути постійним стільки часу, скільки пристрій знаходиться у певному стані певний проміжок часу (реалізація функції затримок описана нижче). В реальній роботі пристрою користувач не знає в який момент часу він натискає кнопку. Це може бути натискання кнопки за достатньо великий проміжок часу до переходу у наступний стан або за декілька миттєвостей до початку роботу функції затримок. В першому випадку, якщо пристрій знаходиться у певному стані 30 секунд, а користувач натискає кнопку на 10 секунді роботи, то кнопка повинна бути затиснута протягом 20 секунд. В другому випадку система коректно відпрацює невеликий проміжок часу натискання кнопки та перейде у інший стан. У випадку з реалізацією обробника переривань від натискання кнопки така неконтрольована ситуація не виникає.

Лістинг 3.5 – Обробник переривань натискання BUTTON_ON та BTN

```
//Pushing button interrupt handler(callback)
//
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	if(GPIO_Pin == GPIO_PIN_1 || GPIO_Pin == GPIO_PIN_2) //Check the
                                    correct pin (btn pin) for insurance
	{
		HAL_TIM_Base_Start_IT(&htim4); //Allow the timer working for
                                                buttons
	}
}
```

Після обробки переривання від натискання кнопки запускається таймер, який рахує час натискання кнопки (за затримкою T_b, яка дорівнює 101 мс). Після закінчення цього проміжку часу виникає подія, яка сигналізує МК щодо

переповнення таймеру та зініціюється переривання від даного таймеру. В обробнику переривання перевіряється чи дійсно кнопка натиснута і тільки після цієї перевірки (якщо кнопка натиснута) змінна стану кнопки зберігає позитивний результат перевірки та дає змогу пристрою перейти у наступний стан. Якщо кнопка не натиснута – пристрій залишається у цьому стані до наступного моменту натискання.

Лістинг 3.6 – Обробник переривання від TIM4

```
//Timer interrupt handler(callback)
//
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	//Delays function
	//
	…
	//Interrupt handler for BUTTON_ON and BTN
	//
	else if(htim->Instance == TIM4)
	{
		if(BUTTON_ON == ON)
		{
		isButtonOnPushed = !isButtonOnPushed; //Change the value of
                                                pushing (like a toggle)
		}
		else if (BTN == ON)
		{
			isBtnPushed = 1;
		}
		else
		{
			isButtonOnPushed = 0;
			isBtnPushed = 0;
		}

		HAL_TIM_Base_Stop_IT(&htim4); //Stop the timer work for
                                                buttons
	}
}
```

Таким чином, така схема вкладеності переривань є типовою схемою реалізації обробки подій за певний проміжок часу.

При переході у стан a1 та натискання кнопки BTN система переходить у стан a2, вихідні сигнали якого повинні бути виведені з певною затримкою (RED_LIGHT та LIGHTER повинні мати «1» на виході у другій половині

роботи стану (від 5 до 10 секунди включно)). Оскільки знаходження системи у стані дорівнює сумі затримок (10 секунд), то необхідно використати деяку умову для відстеження скільки часу система вже знаходиться у цьому стані. Для цього необхідно використати дані, які знаходяться у регістрі таймеру функції затримок (TIM3-регістр CNT). Таким чином і реалізується одна із затримок в системах реального часу – затримка вихідних значень (output delay).

Графічне відображення затримки вихідних значень зображено на рис.3.10.

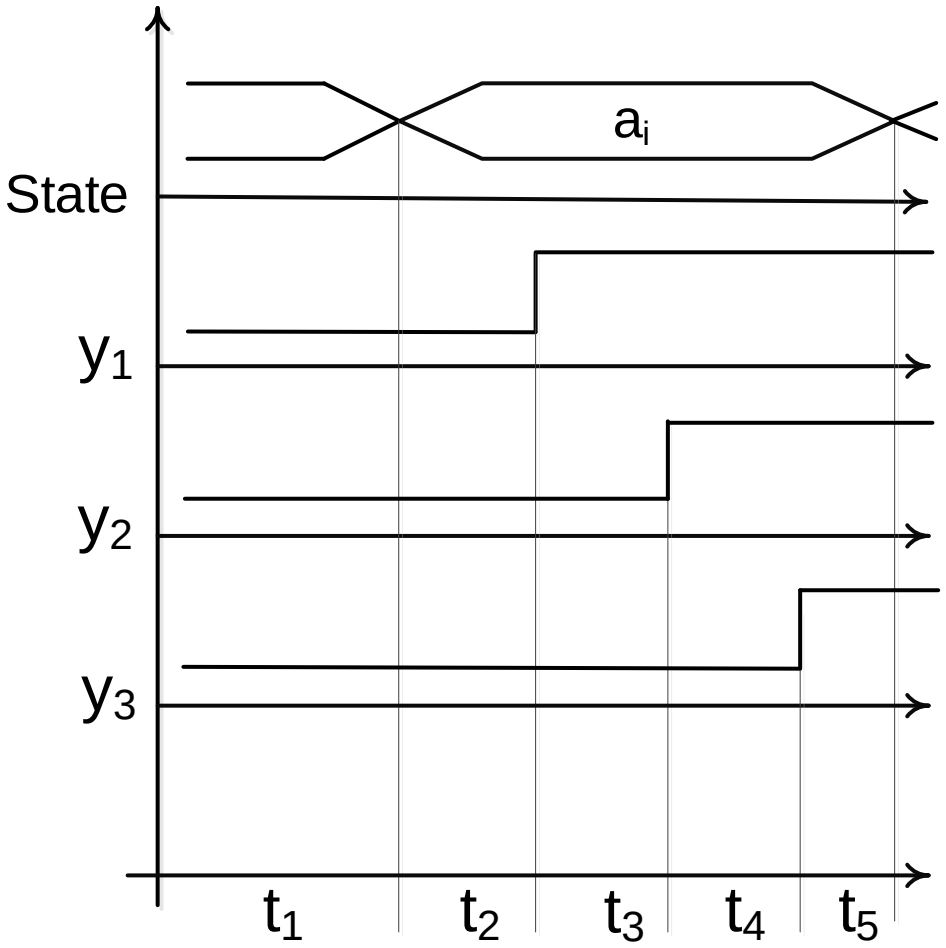

Рисунок 3.10 – Приклад часової діаграми для відображення роботи затримки на виході

Лістинг 3.7 – Реалізація output delay

```
switch(state)                          /*Outputs function*/
{
      …
```

```
            case a2:

                 //Output delay that allows to operate with a red light and
                      lighter at the certain time
                 //ON the red light and the lighter when the state time
                      between 5 and 10 seconds of state
                 //
                 if(__HAL_TIM_GET_COUNTER(&htim3) > delays[T3])
                 {
                      isBtnPushed = 0;
                      HAL_GPIO_WritePin(LEDS,RED_LIGHT|LIGHTER,ON);
                 }
                break;
        …
}
```

Після відпрацювання стану a2 система переходить у гіперпроцес, який відповідає за підсвічування смуг, що сигналізує про дозвіл переходу. Гіперпроцес без умов є нескінченим, але як правило повинна існувати найголовніша умова виходу з гіперпроцесу. Це є час, який система повинна знаходитись у ньому. При завершені часу, виділеного для гіперпроцесу система автоматично повинна вийти з нього та перейти до наступного стану. Оскільки гіперпроцес складається зі станів, які називаються гіперстанами, то виникає ситуація, коли необхідно самостійно зробити підсумок знаходження системи у гіперпроцесі за сумою всіх затримок станів. Також гіперпроцес повинен бути замкнутим, але тільки перший та останній стан виконують цю умову. Інакше виникає неможливість реалізації переходу із стану в стан, тому що такої умови в принципі не може бути (рис. 3.11).

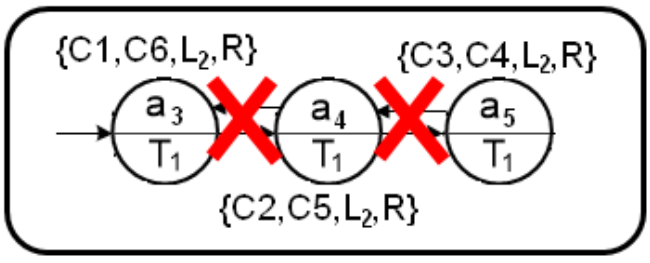

Рисунок 3.11 – Абсурдна ситуація у гіперпроцесі

Тому у гіперпроцесі виникає дублювання станів (a4 та a6 за логікою однакові), але це запобігає порушенню умови створення та реалізації гіперпроцесу.

Лістинг 3.8 – Реалізація гіперпроцесу

```
//Transitions function
//
switch(state)
{
      case a3:

            if(walkTime >= delays[T1]*4) /*Check the time in hyper process and select the right next state*/
            {
                  nextState = a7;
            }

            else
            {
                  nextState = a4;
            }

      break;

      case a4:
            nextState = a5;
      break;

      case a5:
            nextState = a6;
      break;

      case a6:
            nextState = a3;
      break;
      …
}


switch(state)                       /*Outputs function*/
{
      …
      /*Hyper States outputs blocks*/
      case a3:
             HAL_GPIO_WritePin(LEDS,C1|C6,ON);
             HAL_GPIO_WritePin(LEDS,C2|C5,OFF);
             HAL_GPIO_WritePin(LEDS,C3|C4,OFF);
      break;

      case a4:
```

```
                HAL_GPIO_WritePin(LEDS,C1|C6,OFF);
                HAL_GPIO_WritePin(LEDS,C2|C5,ON);
                HAL_GPIO_WritePin(LEDS,C3|C4,OFF);
        break;

        case a5:
                HAL_GPIO_WritePin(LEDS,C1|C6,OFF);
                HAL_GPIO_WritePin(LEDS,C2|C5,OFF);
                HAL_GPIO_WritePin(LEDS,C3|C4,ON);
        break;

        case a6:
                HAL_GPIO_WritePin(LEDS,C1|C6,OFF);
                HAL_GPIO_WritePin(LEDS,C2|C5,ON);
                HAL_GPIO_WritePin(LEDS,C3|C4,OFF);
        break;
        /*END Hyper States outputs blocks*/
        …
}
```

Після завершення гіперпроцесу система переходить у стан завершення переходу, вимкнення червоного світла (RED_LIGHT) та ліхтаря переходу (LIGHTER) та відкриття певного часового вікна на реакцію на зовнішню подію натискання кнопки для ініціації переходу (початок повного обходу по графу, тобто повний цикл робот системи). Таким функціоналом наділено стан a7, який описує декілька затримок автоматного шаблону. Вимкнення світла та ліхтаря має відбутися протягом затримки $T_{3.}$ Інший час затримки у стані – це вікно обмежень (timing constraint) $T_{C.}$ Протягом всієї роботи стану з такою затримкою дозволяє користувачу знову зініціювати перехід, але з умовою того, що дане вікно обмежень не має перериватися та користувач повинен дочекатись, коли почнеться підсвічення смуг. Час роботи стану a7 так як і час роботи стану a2 складається із суми двох затримок. Тому, аналогічно зі станом a2, значення на виході також мають бути перевірені за часом, який лишилося системі провести у стані a7.

Як зазначено вище, у стані a7 система готова до прийняття переривання від зовнішньої події. Тому цей стан також має повну копію функціоналу стану, але з відмінністю у часі, який дає змогу відновити автомобільний рух на деякий час та не залишити пішоходів у подиві чому не підсвічується кнопка здійснення переходу (BTN_LIGHT).

Лістинг 3.9 – Опис останнього стану з затримкою на виході та вікном обмежень

```
void StateTransition()
{
	…
	switch(state)                        /*Outputs function*/
	{
		case a7:

			HAL_GPIO_WritePin(LEDS,C1|C6,OFF);

			if(__HAL_TIM_GET_COUNTER(&htim3) > delays[T3])
			{
				HAL_GPIO_WritePin(LEDS,RED_LIGHT|LIGHTER,OFF);

				//COPY of a1 state that shows the ability to push the btn in a constraint time to the next state transition
				//
				if(isBtnPushed == 0)
				{
					BTN_READY;
					HAL_GPIO_WritePin(LEDS,BTN_LIGHT,ON);
				}
				else
				{
					BTN_NOT_READY;
					HAL_GPIO_WritePin(LEDS,BTN_LIGHT,OFF);
				}
			}

		break;
	…
	}
}
```

Робота пристрою неможлива без опису функції затримок. Так як саме ця функція ініціює перехід із поточного стану в наступний. При цьому встановлює затримку у стані, вносить її до регістру ARR таймеру TIM3 та знову оповіщає систему про переривання від таймеру за переповненням. При цьому програмно змінюється налаштування за замовчуванням при підключенні таймеру у CubeMX.

Також функція затримок взаємодіє зі змінною яка підраховує час перебування системи у гіперпроцесі.

Програмна реалізація функції затримки – обробник переривання від таймеру.

Лістинг 3.10 – Функція затримок

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

			uint16_t delayMs = 0; //Variable for keeping the chosen delay

			state = nextState;	//Transition execution

			if(state == a0 || state == a1)
			{
				delayMs = delays[T0];
			}

			else if(state == a2)
			{
				delayMs = delays[T2]+delays[T3];
			}

			else if(state == a3 || state == a4 || state == a5
	|| state == a6)
			{
				delayMs = delays[T1];
				walkTime += delays[T1];
			}

			else if(state == a7)
			{
				delayMs = delays[T3]+delays[Tc];
				walkTime = 0;
			}

			else //for insurance to avoid the  exceptions
			{
				delayMs = delays[T0];
			}

			__HAL_TIM_SET_AUTORELOAD(&htim3, delayMs); //Reload the
necessary timeout to the timer register

			//The correct way in interrupt handling
			//is the stop and start the timer again (including interrupt
```

```
            //handling)
            //
            HAL_TIM_Base_Start_IT(&htim3);
        }
        …
}
```

Істинний обробник переривань від таймеру або натискання кнопки має суфікс _IRQHandler та створюється у файлі опису всіх обробників (<назва МК>_it.c). Гарним тоном програмування пристроїв на мікроконтролерах є визначення функції зворотного виклику, основна задача якої і є опис логіки обробника подій після того як система отримала сигнал щодо початку обробки переривання, заборонила реакцію на подію, доки не закінчиться обробка поточного переривання та сповістила систему про успішне виконання дій з відповідними регістрами контролера переривань з відповідністю до поточного переривання.

Лістинг 3.11 – приклад обробника переривань (файл stm32f4xx_it.c)

```
void EXTI1_IRQHandler(void) //IRQ handler for BUTTON_ON
{
  /* USER CODE BEGIN EXTI1_IRQn 0 */

  /* USER CODE END EXTI1_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);//All necessary system operations
                                       with interrupt registers
  /* USER CODE BEGIN EXTI1_IRQn 1 */

  /* USER CODE END EXTI1_IRQn 1 */
}
```

Після виконання дій з відповідними регістрами контролера переривань, визивається функція зворотного виклику. Ця функція реалізується автоматично, але має специфікатор __weak, який інформує компілятор про те, що функція може бути реалізована розробником і при виклику цієї функції будуть виконуватися операції тіла перевизначеної функції.

Функції зворотного зв’язку спрощують читабельність коду та роблять його дещо компактним, так як весь реалізований функціонал зводиться в 1-2 файла (файлу заголовків та файл опису), а опис переривань зводиться в одну

функцію з розділенням функціоналу за умовою на якому піні або в якій периферії відбулася подія. У вигляді функцій зворотного зв'язку реалізовано функції затримок та переривання від кнопок.

3.4 Альтернативний опис пристрою за концепцією автоматного шаблону

Однак як завжди існують деякі альтернативи. З приводу реалізації автоматного шаблону на МК можна відмовитись від деяких принципів, вигравши на цьому деяку кількість пам'яті чи зменшити втручання у системну частину МК. Принцип дії пристрою за альтернативним сценарієм роботи можна відтворити за допомогою графу переходу, зображеного на рис.3.12.

Головною відмінністю моделі керуючого автомату є виключення затримки на виході. На перший погляд умова зчитування поточного часу до переривання від таймеру досить проста (лістинг 3.8), але специфічна, так як при реалізації автоматного шаблону на МК виникає ситуація коли постійно зчитуються дані з регістру CNT таймеру TIM3 (взаємодія з системними параметрами), навіть якщо це нам не потрібно, так як функція виконується безперервно до моменту переривання від переповнення таймеру TIM3, тобто початку роботи функції затримок. До того ж такий спосіб дуже рідко зустрічається у реальних проектах.

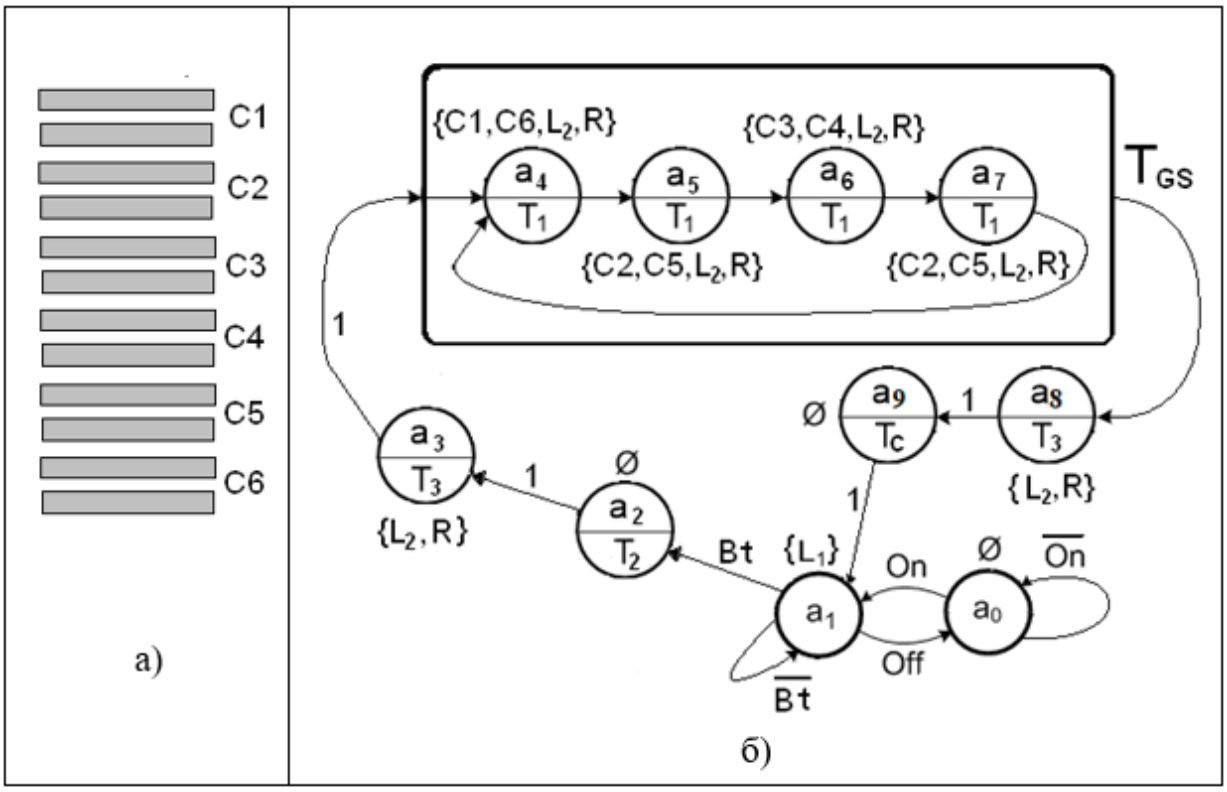

Рисунок 3.12 – Модель керуючого автомату освітлення пішохідного переходу за альтернативним описом АШ на МК: а) вихідні сигнали підсвічення смуг; б) граф переходів

З одного боку така реалізація дещо порушує принцип автоматного шаблону опису системи реального часу. Але з іншого боку виникає більш важлива риса таких систем – розділення відповідальності. Тобто кожний стан можна розглянути як певну задачу, яку виконує пристрій. За правилом одна задача повинна виконувати одну певну дію або декілька дій, пов'язаних між собою. Наприклад одна задача відповідає за зчитування даних та їх відправлення до другої задачі. Друга задача, на основі зчитаних даних, формує графік та передає дані графіку до третьої задачі. Третя задача виводить цей графік на дисплей. Таким способом реалізується механізм операційної системи реального часу (ОСРЧ) – тип операційної системи, основне призначення якої – надання необхідного та достатнього набору функцій для роботи систем реального часу на конкретному апаратному обладнанні [20]. Однак, розглядаючи реалізацію ОСРЧ у МК, необхідно повністю

реорганізувати автоматний шаблон за стилем його опису та особливостями проектування самої операційної системи.

Таким чином, можна стверджувати, що даний сценарій поєднує в собі частково логіку ОСРЧ та стандартного опису автоматного шаблону.

Однак варто зауважити, що даний опис, дивлячись на механізм розділення відповідальності, має більше станів ніж стандартний шаблон. Такий механізм називається розщепленням станів. Саме він дозволяє використати частину логіки ОСРЧ.

Для порівняння двох шаблонів, що розглядаються, всі переваги та недоліки наведені у табл.3.1

Інколи виникає ситуація, коли стан може не мати певних дій, але за логікою виключення затримок на виході повинно бути реалізоване (кожний стан, який має затримку на виході повинен бути розщеплений). Такі стани називаються «холостими». Їх основна задача – затримати час включення певних сигналів у наступному стані. Про це також зазначено у табл.3.1.

Механізм розщеплення станів відображено на рис.3.13

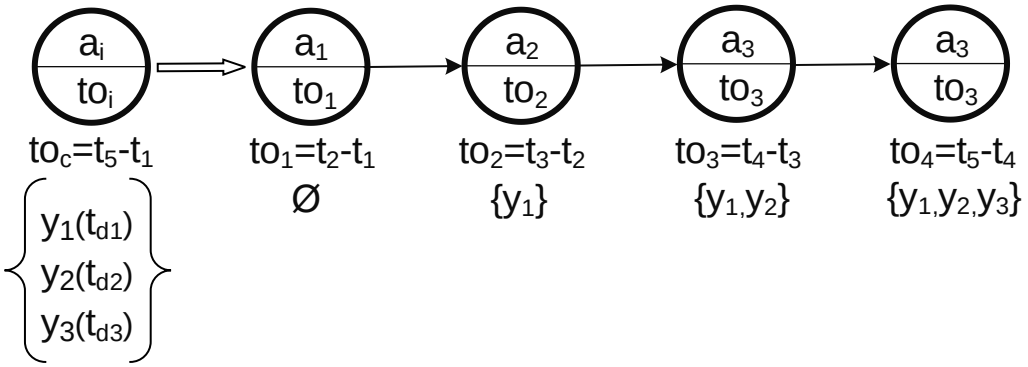

Рисунок 3.13 – Механізм розщеплення станів

Такий механізм змінює часову діаграму роботи затримки на виході зображену на рис.3.14.

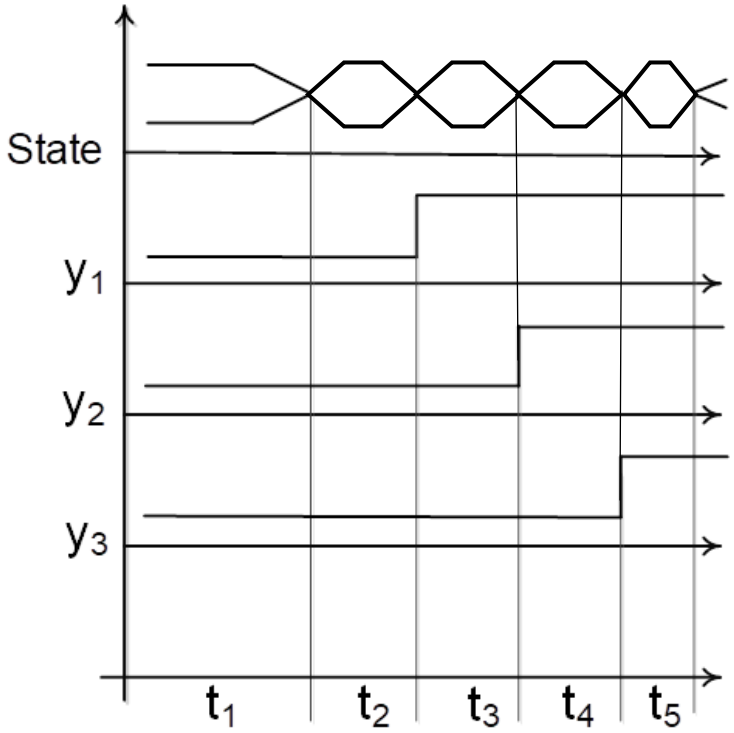

Рисунок 3.14 – Часова діаграма роботи механізму розщеплення станів

Реалізацію розщеплення станів продемонстровано у лістингу 3.13.

Лістинг 3.13 – Опис розщеплення станів a2, a3 та a8, a9

```
switch(state)                          /*Outputs function*/
{
      …
      case a2:

            /*
              *An idle state where the outputs may not be described
              *This is a key point of state splitting mechanism to exclude
                  the output delays
              *But for supporting the description of the template, the
                  outputs should be described
              *In this case the BTN and the BTN interrupt are not
                  available
            */
                  isBtnPushed = 0;
                  BTN_NOT_READY;
            break;

            case a3:
             HAL_GPIO_WritePin(LEDS,RED_LIGHT|LIGHTER,ON);
            break;
      …
            //The vivid example of a transition splitting to hide the
```

```
		//output delays meaning (a8 and a9)
		//
		case a8:
			HAL_GPIO_WritePin(LEDS,C1|C6,OFF);
		break;

		case a9:

			HAL_GPIO_WritePin(LEDS,C1|C6,OFF);
			HAL_GPIO_WritePin(LEDS,RED_LIGHT|LIGHTER,OFF);

			//COPY of a1 state that shows the ability to push the
			//BTN in a constraint time to the next state transition
			//
			if(isBtnPushed == 0)
			{
				BTN_READY;
				HAL_GPIO_WritePin(LEDS,BTN_LIGHT,ON);
			}
			else
			{
				BTN_NOT_READY;
				HAL_GPIO_WritePin(LEDS,BTN_LIGHT,OFF);
			}

		break;
	…
}
```

В функції переходів реалізується безумовний перехід із стану в стан.

Функція затримок також змінюється числом станів. Кожному стану призначається своя власна затримка без підсумовування та інших арифметичних операцій з затримками, які зберігаються у відповідному масиві.

Лістинг 3.14 – Функція затримок при розщеплені станів a2,a3 та a8,a9

```
//Timer interrupt handler(callback)
//
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	//Delays function
	//
	if(htim->Instance == TIM3)
	{
		…
		else if(state == a2)
		{
			delayMs = delays[T2];
		}
		else if(state == a3)
```

```
                {
                        delayMs = delays[T3];
                }
                …
                else if(state == a8)
                {
                        delayMs = delays[T3];
                        walkTime = 0;
                }
                else if(state == a9)
                {
                        delayMs = delays[Tc];
                }
                …
        }
}
```

Окрім цих змін в описі шаблону все інше залишається незмінним.

3.5 Результат роботи пристрою. Порівняння шаблонів

Принцип роботи пристрою графічно можна відобразити часовою діаграмою, зображеною на рис.3.15.

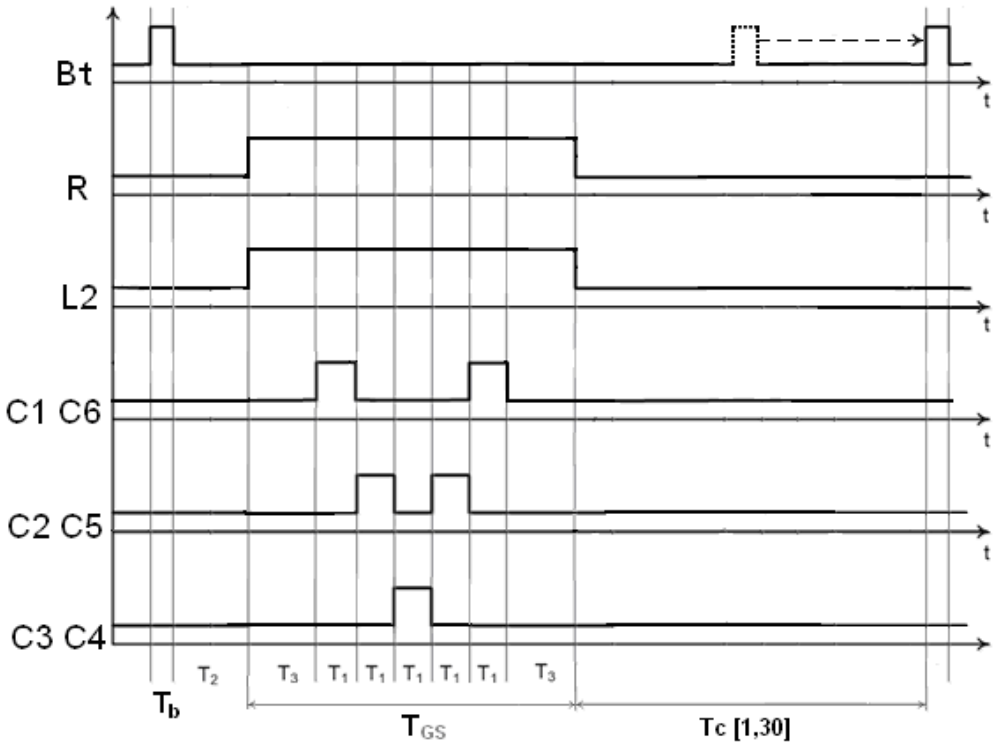

Рисунок 3.15 – Часова діаграма функціонування керуючого автомату освітлення

Результат роботи пристрою за реалізацією двох варіантів автоматного шаблону на МК продемонстровано на часовій діаграмі, яку сформовано у реальному часі, тобто паралельно з роботою пристрою (рис.3.16).

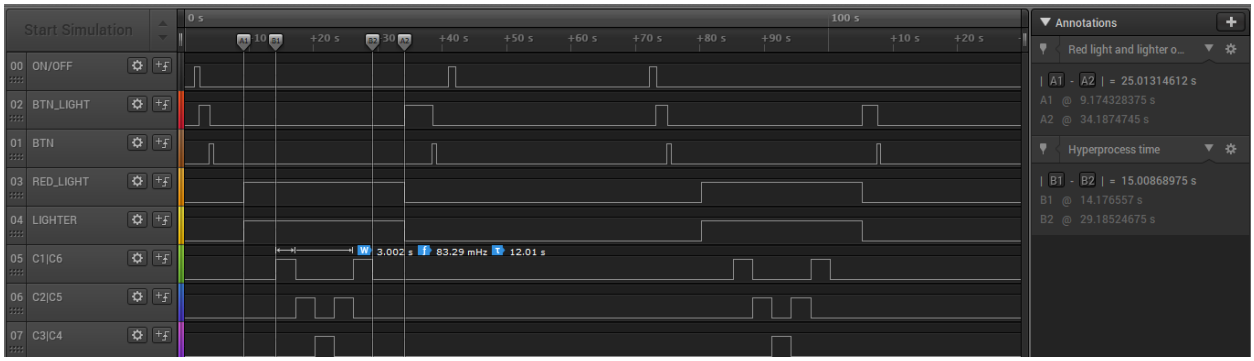

Рисунок 3.16 – Часова діаграма функціонування керуючого автомату освітлення в реальному часі

Часова діаграма, що представлена на рис.3.16 відображає роботу пристрою за повним обходом графу будь-якого автоматного шаблону. Всі часові показники збережені, але мають невелику похибку у більшу сторону. Багато причин впливає на неї. Наприклад, нестабільна подача живлення на пристрій або на логічний аналізатор. Перепад напруги на декілька десятих вольт може призвести до зміни тактової частоти роботи пристрою або логічного аналізатора, що призводить до їх асинхронної роботи. Або відсутність якомога точнішого налаштування періоду таймеру у проекті призводить до збільшення тривалості сигналу для синхронізації із заданою частотою тактового генератора. Існує ще дуже багато причин, тому відстеження та усунення проблеми, яка призводить до настільки малої похибки не розглядається.

Також на часовій діаграмі відображено роботу системи, коли у стані з вікном обмежень була натиснута кнопка ініціації переходу та згодом система була вимкнута. Ця ситуація детальніше описано нижче.

Важливо відмітити, що у програмній реалізації змінні, які зберігають значення поточного та наступного стану внутрішні і не підключені до GPIO.

Окрім того, виведення значень станів на часову діаграму повинно бути реалізовано за допомогою протоколів передачі даних, так як порти являються дискретними і не можуть без додаткової логіки мати інші вихідні сигнали, окрім «0» та «1». Також виведення станів не є задачею системи і відноситься тільки до апаратного відлагодження. До того ж для виведення станів необхідно використовувати логічний аналізатор більше ніж на 8 каналів.

Тому відображення станів скориговано власноруч, згідно з реалізацією автоматного шаблону (стандартного та з механізмом розщеплення станів).

Як описано вище, часова діаграма однакова для всіх реалізованих шаблонів. Всі часові характеристики однакові для кожної реалізації. Основна відмінність полягає у відображенні появи високого або низького рівня сигналу за станами автоматних шаблонів.

Часова діаграма роботи пристрою за стандартним автоматним шаблоном представлена на рис.3.17. Часова діаграма роботи пристрою за альтернативним сценарієм представлена на рис.3.18.

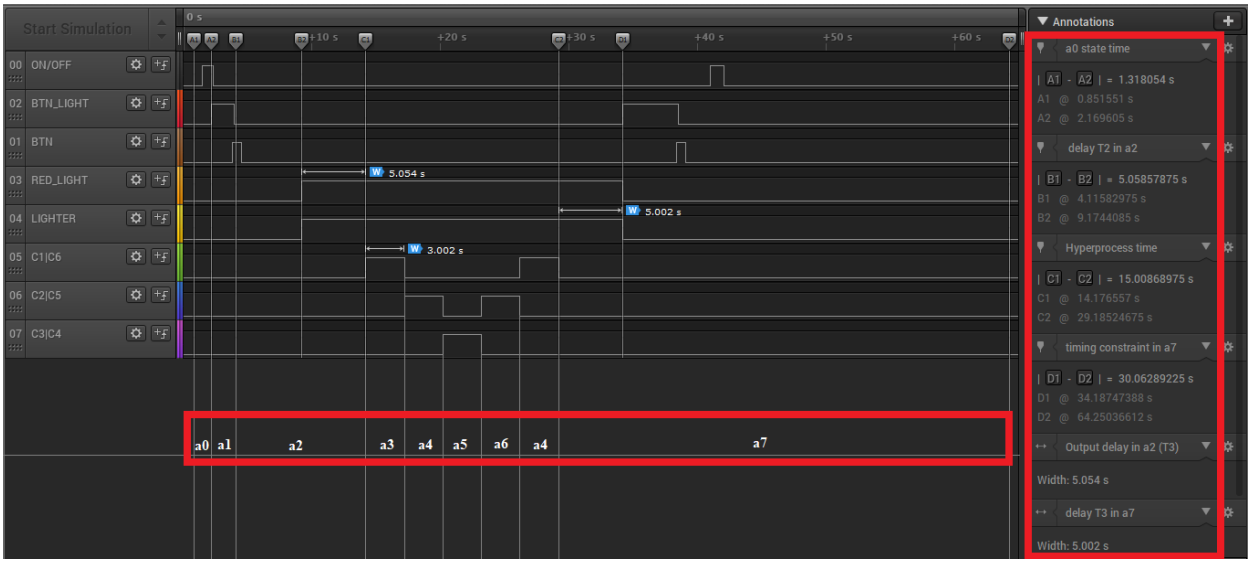

Рисунок 3.17 – Часова діаграма функціонування керуючого автомату освітлення в реальному часі за реалізацією стандартного автоматного шаблону

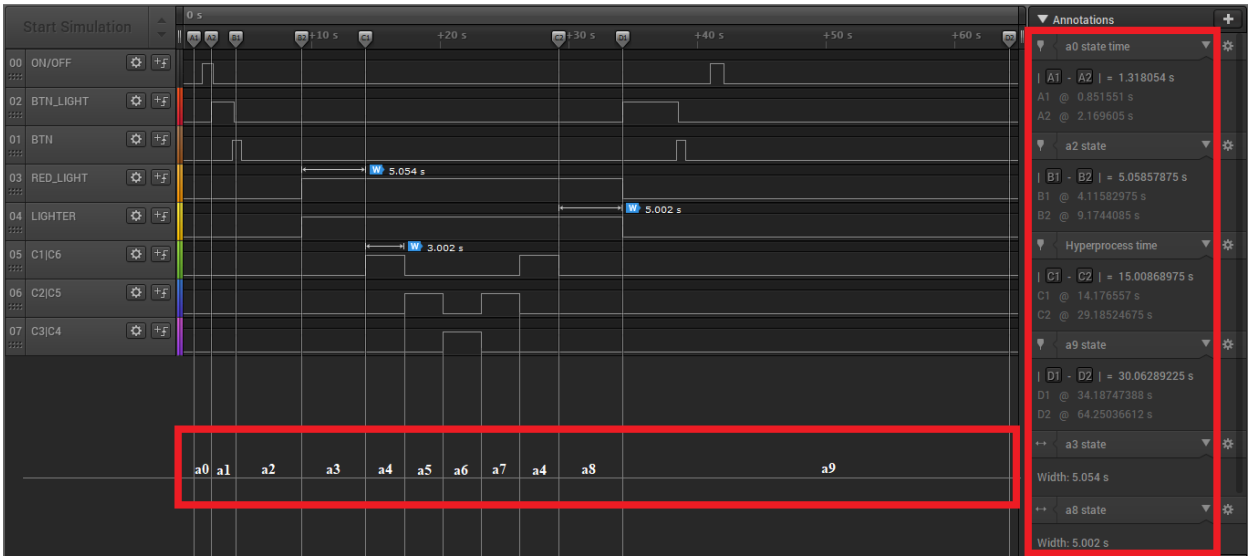

Рисунок 3.18 – Часова діаграма функціонування керуючого автомату освітлення в реальному часі за реалізацією автоматного шаблону з розщепленням станів

Порівняння рис.3.17 та рис.3.18 повністю ще раз підтверджує те, що повний цикл роботи пристрою не змінюється у часі.

Часова діаграма на рис.3.16 має вдвічі більший час повного циклу роботи пристрою. В даній діаграмі розглянуто ситуацію, коли система знаходиться у стані з вікном обмежень. Користувач натискає кнопку ініціації переходу, але реакція системи повинна відбутися тільки після закінчення часу вікна обмежень, переходу у стан a1 в якому і буде перевірено чи була натиснута кнопка. Така поведінка системи дозволяє відновити рух транспорту на деякий час та водночас забороняє користувачу постійно безперервно натискати на кнопку. Водночас відбувається дія, яка зображена на рис.3.15 з «уявним перенесенням» сигналу натискання кнопки ініціації переходу у стан a1.

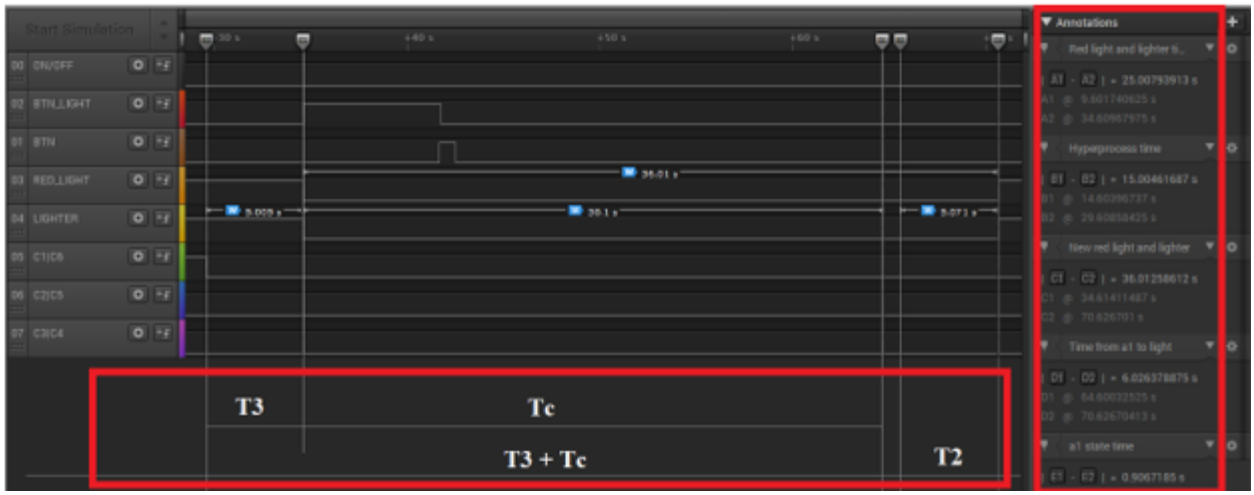

Рисунок 3.19 – Часова діаграма затримок при «уявному перенесенні» сигналу натискання кнопки у стан a1

Варто зазначити, що вимірювання часу затримок є ручною дією, тому можуть бути невеликі похибки у часі. Прикладом цього є час роботи системи у стані a1 (маркери проміжку часу E1 та E2). Оскільки вимірювання вікна обмежень складає 30.1с, а вимірювання затримки T2 становить 5.071с, то, беручи до уваги повний проміжок часу від початку вікна обмежень до увімкнення червоного світла та ліхтаря освітлення пішохідного переходу, з урахуванням всіх похибок, отримаємо чистий час у 0.9с.

Підводячи підсумки реалізації стандартного автоматного шаблону та шаблону з механізмом розщеплення станів необхідно узагальнити всі переваги та недоліки кожного з шаблону та порівняти їх. Порівняння шаблонів за перевагами та недоліками наведено у табл.3.1.

Таблиця 3.1 – Порівняльна характеристика програмної реалізації пристрою

Стандартний шаблон	Альтернативний шаблон (розщеплення станів)
Суворе дотримання реалізації всіх затримок	Виключення поняття затримок вихідних значень
Безперервне опитування системного таймеру на час, який залишився до закінчення певної затримки	Механізм «розщеплення станів»,
Поєднання декількох однотипних задач в одному стані	Реалізація принципу розділення відповідальності
Наявність арифметичних операцій в функції затримок для формування загальної затримки стану	Створення «холостих» станів
Збільшення швидкості обчислення за рахунок затримок вихідних значень	Виключення будь-яких логічних умов та арифметичних операцій в роботі із параметрами часу (зі змінними, що зберігають затримку або з системними параметрами)

ВИСНОВКИ

В ході виконання атестаційної роботи розроблено мікроконтролерний пристрій керування освітленням пішохідного переходу на основі кінцевого автомата. При проведенні розробки докладно розглянуті математичні основи пристроїв логічного управління та при побудові математичної моделі пристрою керування застосована надбудова над стандартною моделлю кінцевого автомата, а саме, гіперавтомат. Ця конструкція дозволяє будувати моделі кінцевих автоматів, у яких перехід між станами залежить виключно від параметру часу знаходження у відповідному автоматному стані. При аналізі вхідних сигналів застосований поділ на вхідні дії («input patterns») та події («events»). Запропонований алгоритм обробки подій, які є зовнішніми по відношенню до сигналів кінцевого автомату. Модель пристрою керування представлена у вигляді темпорального графу переходів керуючого кінцевого автомата Мура.

Розглянуто поняття систем реального часу та часових автоматів. Проаналізовано спосіб їх використання у мікроконтролерних системах. На основі аналізу створено автоматний шаблон для мікроконтролерних систем, який відповідає всім критеріям систем реального часу.

При розробці мікроконролерного пристрою було використано концепцію автоматного шаблону для мікроконтролерних систем. Програмна реалізація має два сценарії (шаблони): стандартний та альтернативний. Стандартний шаблон відповідає всім часовим параметрам, які вкладені у концепцію часових автоматів (реалізовані затримки на виході, затримки станів та вікно обмежень). Альтернативний шаблон виключає поняття затримок на виході, за рахунок механізму розщеплення станів, що призводить до використання концепції розділення відповідальності (кожен стан – певна задача).

Розглянуто та проаналізовано результат роботи у реальному часі за допомогою апаратного відлагодження за допомогою логічного аналізатора. Було виявлено похибки у вимірах часу включення/виключення сигналів, що є результатом певних зовнішніх чинників, які проконтролювати майже нереально (перепади напруги, розсинхронізація мікроконтролерного пристрою та логічного аналізатора, затримка вимірювань через брак оперативної пам'яті ПК при моделюванні).

Результати роботи пристрою за двома сценаріями ідентичні і не мають жодних відхилень. Тому використання того чи іншого сценарію залежить суто від специфіки задач системи або ж специфікації.

Створено порівняльну характеристику реалізованих сценаріїв. Її аналіз демонструє, що стандартний шаблон має більше програмних умов щодо його реалізації (арифметика затримок, додаткове право на доступ до системних регістрів МК), а сценарій з «розщепленням станів» демонструє збільшення станів керуючого автомата при зменшенні логічних умов та арифметичних операцій з параметрами часу.

Мікроконтролерний пристрій керування освітленням пішохідного переходу є системою логічного управління. Оскільки основна місія такої системи – це управління дискретними сигналам на вході та виході, то доцільно стверджувати, що додаткова периферія (наприклад додаткова кількість смуг або пристрій звукового супроводу пішохода) не ускладнює розробку системи у програмній реалізації. Всі операції зводяться до, можливо, створення ще одного стану, в якому може знаходитися система, включення ще одного або декілька пінів для зв'язку у налаштуванні проекту та додатковим присвоєнням значень на виході при описі гіперпроцесу.

Реалізація даної системи повністю підтверджує сучасність розглянутих методів опису пристроїв. Використання таких методів повністю може використовуватися при розробці різних проектів, роль яких становить управління сигналами у фіксовані моменти часу та підтримкою системи обробки подій та переривань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мисюк Ю.П., Зовнішнє освітлення міст та безпека дорожнього руху / Ю. П. Мисюк // Світлотехніка та електроенергетика. – 2010. – № 3-4. –С. 33-39.

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

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

4. Гилл А.. Введение в теорию конечных автоматов /Артур Гилл. – М.: Издательство «Наука», 1966. – 272 с.

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

6. Миллер Р. Теория переключательных схем, т. II. Последовательностные схемы и машины: перев. с англ. / Реймонд Е. Миллер. – М.: Издательство «Наука», 1971. – 304 с.

7. Орлов С.А. Теория и практика языков программирования: учебник для вузов. Стандарт 3-го поколения. – СПб.: Питер, 2013. – 688 с.

8. Матюшин А.О. Программирование микроконтроллеров: Стратегия и тактика / А.О. Матюшин. – М.: ДМК Пресс, 2017. – 356 с.

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

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

11. Merayo M.G. Formal Testing from Timed Finite State Machines / M.G. Merayo, M. Nunez, I. Rodriguez // Computer Networks. – 2008. – Vol. 52– №2. – P. 432-460.

12. Зюбин В.Е. Программирование информационно-управляющих систем на основе конечных автоматов: учеб.-метод. пособие / В.Е. Зюбин. –

Новосибирск: Новосиб. гос. ун-т., 2006. – 96 с.

13. Карпов Ю.Г. Теория автоматов / Ю.Г. Карпов. – СПб.: Питер, 2003. – 208 с.

14. STM32F427xx STM32F429xx Datasheet – production data [Електронний ресурс]/ST Life augmented – Режим доступу: www / URL: https://www.st.com/resource/en/datasheet/stm32f429zi.pdf – 10.12.2019р. – Загол. З екрану

15. STM32Cube initialization code generator [Електронний ресурс] / STM32CubeMX – Режим доступу: www / URL: https://www.st.com/en/development-tools/stm32cubemx.html – 10.12.2019 г.– Загол. з екрану.

16. Бітченко О.М. Електроніка і мікросхемотехніка. Проектування і програмування мікропроцесорних пристроїв: підручник / О.М. Бітченко, О.І. Цопа, Д.Г. Ганшин. – Х.: Фінарт, 2016. – 354 с.

17. Description of STM32F4 HAL and LL drivers [Електронний ресурс] / UM1725 User Manual – Режим доступу: www / URL: https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf – 10.12.2019 г.– Загол. з екрану.

18. Logic Analyzer [Електронний ресурс] / Wikipedia The Free Encyclopedia – Режим доступу: www / URL: https://en.wikipedia.org/wiki/Logic_analyzer– 10.12.2019 г.– Загол. з екрану.

19. Software development tool suite for Arm Cortex-M based microcontrollers. [Електронний ресурс] / ARM developer. Keil MDK – Режим доступу: www / URL: https://developer.arm.com/tools-and-software/embedded/keil-mdk – 10.12.2019 г.– Загол. з екрану.

20. Операційна система реального часу [Електронний ресурс] / Wikipedia The Free Encyclopedia – Режим доступу: www / URL: https://uk.wikipedia.org/wiki/Операційна_система_реального_часу – 10.12.2019 г.– Загол. з екрану

21. Shkil A. Design of Logical Control Units Based on Finite State Machines' Patterns/ M Miroshnyk; S. Poroshyn; A. Shkil; E. Kulak; I. Filippenko; D. Kucherenko; Y. Pa khomov; J. Salfetnikova; M. Goga//Proceedings of the 2018 IEEE East-West Design & Test Symposium. IEEE Xplore Digital Library. https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?filter=issueId%20EQ%20%228524135%22&refinements=Author:Alexander%20Shkil&pageNumber=1&resultAction=REFINE