Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет ___Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра _____Автоматизації проектування обчислювальної техніки_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
(рівень вищої освіти)

_____Керування рухомими об'єктами на основі операційної системи _____.
_____реального часу_____.
_____.
(тема)

Виконав: студент 2 курсу, групи СКСм-18-1
_____Семенцов Д.О._____
(прізвище, ініціали)
Спеціальність _123 – Комп'ютерна інженерія_
(код і повна назва спеціальності)
Тип програми__Освітньо-професійна__
_____
(освітньо-професійна або освітньо-наукова)
Освітня програма_____
_____Спеціалізовані комп'ютерні системи____
_____
(повна назва освітньої програми)
Керівник ___доц. Філіппенко І.В._____
(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
Тип програми Освітньо-професійна
Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

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

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Семенцову Данілу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Керування рухомими об'єктами на основі операційної системи реального часу

затверджена наказом по університету від 04 листопада 2019 р. № 1624 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 12 грудня 2019 р.
3. Вихідні дані до роботи
багатозадачність
Arduino
рухомий об'єкт
сенсорні пристрої
4. Перелік питань, що потрібно опрацювати в роботі
операційна система реального часу
модель рухомого об'єкту
вибір сенсорів
функціональна схема програми управління

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

12 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	Отримання завдання	03.09.2019-07.09.2019	
2	Аналіз предметної області	11.09.2019-21.09.2019	
3	Аналіз джерел з проблемної галузі	25.09.2019-05.10.2019	
4	Розробка моделі пристою керування	08.10.2019-19.10.2019	
5	Розробка моделі пристрою та апаратної частини	22.10.2019-02.11.2019	
6	Розробка програмної частини	05.11.2019-16.11.2019	
7	Проведення тестування	19.11.2019-30.11.2019	
8	Оформлення пояснювальної записки	03.12.2019-14.12.2019	
9	Оформлення графічного матеріалу	17.12.2019-28.12.2019	
10	Перевірка виконаного проекту керівником	02.01.2019-04.01.2019	

Дата видачі завдання вересня 2019 р.

Студент _____
(підпис)

Керівник роботи _____ Філіппенко І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 56 сторінок, 16 рисунків, 1 таблицю, 31 джерело за переліком посилань.

БАГАТОТОЗАДАЧНІСТЬ, СИСТЕМИ РЕАЛЬНОГО ЧАСУ, ОПЕРАЦІЙНА СИСТЕМА

В атестаційній роботі були сформульовані вимоги, які пред'являються до систем реального часу. Були визначені такі поняття як завдання, планувальник і сформульовані вимоги до них. Був проведений аналіз проблеми багатозадачності та способів вирішення цієї проблеми. Був здійснений вибір апаратних пристроїв для реалізації системи. Був проведений огляд існуючих систем реального часу.

Були запропоновані моделі операційної системи, що розробляється, апаратної моделі рухомого об’єкту. Було реалізовано програмні модулі планувальника, модулі управління рухом, опитування датчиків і програмний модуль планувальника. Проведено тестування запропонованого пристрою.

ABSTRACT

Master's thesis contains 55 pages, 16 figures, 1 table, 31 sources according to the list of links.

MULTIPLETASKS, REAL-TIME SYSTEMS, OPERATING SYSTEM

Requirements for real-time systems were formulated in the evaluation work. Concepts such as the task, planner, and requirements have been defined. An analysis of the problem of multitasking and ways of solving this problem was conducted. A selection of hardware devices for the implementation of the system was made. An overview of existing real-time systems was conducted.

Models of the operating system being developed, a moving object hardware model, have been proposed. The scheduler software modules, motion control modules, sensor surveys, and scheduler software modules were implemented. The proposed device was tested.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КА – керуючий автомат;

МК – мікроконтролер;

САПР – система автоматизованого проектування;

ОСРЧ – операційна система реального часу;

ІК – інфрачервоний датчик;

ШИМ – широтно-імпульсна модуляція.

ВСТУП

На сьогоднішній день мікроконтролери використовуються повсюдно для реалізації завдань управління в різних пристроях. Вони призначені для управління різноманітними системами згідно з розробленою мікроконтролерною програмою. Мікроконтролери використовуються як в системах побутового так промислового призначення. При цьому кількість задач, які мікроконтролер повинен вирішувати паралельно, постійно збільшується. Виходячи з цього, мікроконтролерні програми стають все складніше як в процесі написання, так і налагодження. Часто виникає проблема багатозадачності, яка повинна бути вирішена за допомогою розробки управляючої програми. При розробці практично будь-якого програмного забезпечення для мікроконтролерів з'ясовується, що програма має складатися з декількох, порівняно самостійних завдань з можливістю комунікації цих задач між собою. Тобто існує потреба в загальній керуючій програмі названою операційною системою, що повинна працювати в реальному масштабі часу [1].

На сьогоднішній день отримали розвиток 8-ми і 32-х розрядні мікроконтролери. Для вирішення завдань багатопоточності в мікроконтролерах можливе використання готових операційних систем реального часу, які представлені на ринку на даний момент. Однак не завжди готові рішення можуть відповідати вимогам системи, яка розробляється. Тоді виникає необхідність розробки власної операційної системи реального часу самостійно.

У свою чергу, написання програм для мікроконтролерів різко відрізняється від написання програм для універсального комп'ютера. При виконанні програми на комп'ютері запуск програм, взаємодія з внутрішніми, зовнішніми пристроями або людиною бере на себе операційна система. Програма, написана для мікроконтролера, повинна вирішувати всі ці задачі

сама. Прикладом багатозадачності може виступати задача опитування датчика, виведення даних на екран і паралельне опитування клавіатури.

Виконувана програма являє собою чітку послідовність дій. І тільки від реалізації управляючої програми залежить вирішення проблеми багатозадачності, тобто яка послідовність буде виконуватися в даний момент і що вона робитиме в кожен конкретний момент часу. Не дивлячись на велику різноманітність запропонованих варіантів рішення, проблема реалізації багатозадачності на мікроконтролерах на сьогоднішній день залишається актуальною.

Метою атестаційної роботи є розробка програми управління рухом об'єкта на підставі операційної системи реального часу. Об'єктом дослідження є мобільні рухливі об'єкти. Предметом – операційні системи реального часу для управління мобільним рухомим об'єктом.

1 ОПЕРАЦІЙНІ СИСТЕМИ РЕАЛЬНОГО ЧАСУ

1.1 Поняття операційної системи і її архітектура

При розробці систем на мікроконтролерах доводиться вирішувати проблему розподілу часу між задачами, що виконуються, так як майже кожна вбудована система має більш ніж одну задачу, яку необхідно обробляти в реальному масштабі часу. Отже, це системи, які повинні працювати при жорстких часових обмеженнях.

Програма управління об'єктом, що рухається може включати в себе такі задачі, як визначення відстані до об'єктів, освітленості, вимір відстані на яке об'єкт рухається, управління рухом і т.д.

Чим точніше повинен рухатися об'єкт щодо різних перешкод, тим більша кількість завдань, таких як опитування різних датчиків, доводиться вирішувати і програма управління все більше ускладнюється. Стає все важче збалансувати послідовність дій, які система повинна виконувати і все складніше визначати порядок виконання послідовності без шкоди для інших задач щодо один одного. Кожна задача (опитування різних датчиків, управління обертанням моторами, розрахунок траєкторії руху) буде конкурувати за процесорний час і, отже, може викликати затримки в обробці інших задач системи, що миттєво позначиться на якості руху. У цьому випадку об'єкт, що рухається, змушений буде весь час зупинятися на час обробки даних, отриманих від кожного датчика, або весь час буде наїжджати на перешкоди.

Отже, така система повинна обробляти безліч незалежних потоків вхідних даних від різних джерел і на їх підставі виробляти управляючі сигнали. При цьому неможливо чітко передбачити час надходження даних від кожного з датчиків, проте необхідно відреагувати на кожну подію, при

цьому не порушуючи часові обмеження, сформульовані в технічному завданні.

При збільшенні кількості датчиків, можливість появи даних одночасно від декількох джерел, сильно зростає, відповідно зростає ймовірність колізії находження сигналів, що потрібно обробляти. Код програми управління при цьому починає ускладнюватися. Один таймер починає відповідати за виконання кількох періодичних задач одночасно. А переривання – мультиплексируються, в тому числі за допомогою додаткових апаратних блоків. Також виникають такі проблеми: обмежений обсяг пам'яті мікроконтролера, відсутність компактної операційної системи, відсутність чіткого регламенту обробки переривань і критичних ділянок програми.

Всі ці питання можна вирішити за допомогою операційної системи реального часу.

Операційні системи реального часу ОСРВ (англ. Real-Time Operating System) забезпечують багатозадачність системи управління за допомогою сервісів, що надаються ядром, що помітно спрощує розробку систем управління та забезпечує раціональне використання ресурсів процесора [1].

На рисунку 1.1 наведено приклад такої системи реального часу, яка повинна обробляти події від декількох датчиків одночасно.

Важливим паралельним аспектом необхідності спільного використання ресурсів мікропроцесора є необхідність забезпечення того, щоб все відбувалося в реальному масштабі часу. У всіх вбудованих системах виникає наступна проблема, чим більше конкуруючих завдань необхідно вирішувати за допомогою керуючої програми, тим складніше розділяти процесорний час [2].

Виходячи зі сказаного основна вимога, яка висувається до операційної системи реального часу це – забезпечення передбачуваності або детермінованості поведінки системи при будь-яких зовнішніх критичних впливах. Саме ця вимога відрізняє ОСРЧ від вимог, що пред'являються до продуктивності і швидкодії універсальних ОС [4].

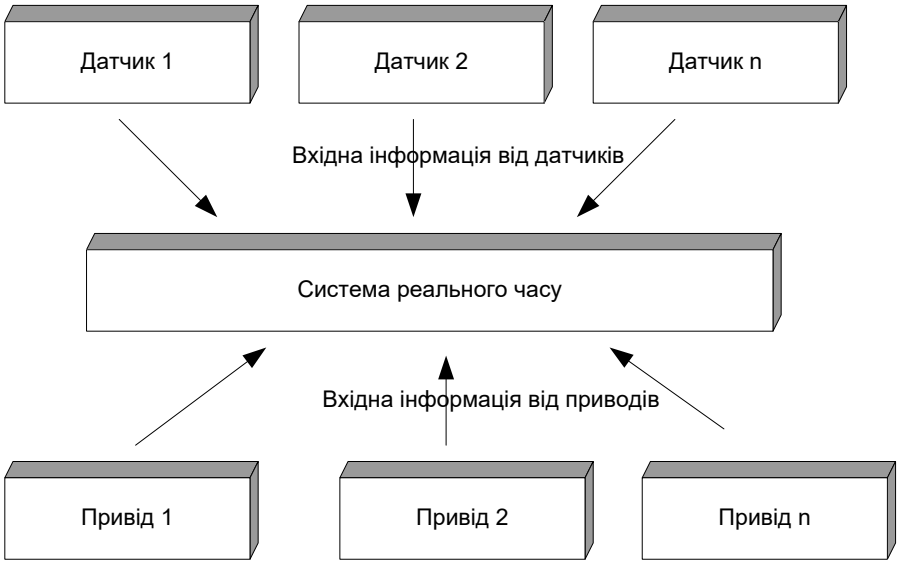

Рисунок 1.1 – Система реального часу

1.2 Робота системи в реальному масштабі часу

Розглянемо визначення: «робота системи в реальному масштабі часу» для виділення ключових моментів при проектуванні ОСРЧ.

Дональд Гілліес сформулював таке визначення роботи системи в реальному масштабі часу: «Системою реального часу є така система, коректність функціонування якої визначається не тільки коректністю виконання обчислень, але і часом, в яке отримано необхідний результат. Якщо вимоги по часу не виконуються, то вважається, що відбулася відмова системи »[4].

Інакше це можна трактувати наступним чином, що система реального часу перебуває в жорстких часових рамках. При цьому часові рамки можуть бути досить великими, так і дуже маленькими, тобто мати на увазі високу швидкість обробки інформації. При цьому якщо обробка вхідних сигналів і вихідна реакція системи задовольняє технічним завданням, то вважається що система повністю відповідає вимогам. Якщо ж часові рамки порушені, то вважається, що система дала неприпустимий збій.

Виходячи зі сказаного, швидкість реакції системи важлива тільки щодо швидкості протікання зовнішніх подій, які система реального часу повинна відстежувати і виробляти відповідні управляючи дії. В цьому випадку можна говорити про коректність роботи спроектованої системи.

Системи реального часу можна розділити на два класи: системи «жорсткого» та «м'якого» часового обмеження [4].

Жорсткі системи реального часу зобов'язані реагувати на подію, що надійшла, в жорстких часових межах, інакше в системі можливий аварійний збій.

Системи м'якого часового обмеження можуть мати запас часу при формуванні відповідної реакції на систему. Вони відрізняються від жорстких тим, що затримка реакції може бути отримана з невеликим запізненням і це не призведе до збою, а тільки до зниження продуктивності системи в цілому. При цьому немає чіткого критерію, на який проміжок часу може затягнутися відповідь системи на отриманий сигнал.

Основна відмінність між системами жорсткого і м'якого реального часу можна виразити так: система жорсткого реального часу не може будь-коли запізнюватися з реакцією на подію, система м'якого реального часу – має можливість спізнюватися з реакцією на подію.

1.3 Задачі. Пріоритети задач

Під задачею будемо розуміти послідовність операцій, які вирішують один певний алгоритм. Тобто іншими словами це частина коду, яка з точки зору програміста повинна виконуватися «одночасно» з іншими задачами і при цьому може функціонувати незалежно від інших шматків коду.

Прикладом задач можуть служити, наприклад, опитування датчиків, опитування клавіатури або вивід на екран якихось даних.

Задачі можуть виконуватися просто в суворій послідовності. При цьому задачі виконуються послідовно один за одним, без переривань, тоді

проста програма задовольняє цим умовам, використовуючи цикли, умовні переходи і виклики підпрограм.

Кожна задача може мати пріоритет. Розподіл пріоритетів між задачами відбувається конкретно в кожному проекті, що розробляється в залежності від важливості і часових умов технічного завдання.

У мікроконтролерах з малим об'ємом пам'яті програм поняття «задача» та «потік» можна вважати синонімами [2]. При цьому для кожного потоку необхідно визначити наступне: віртуальний адресний простір; виконуючий код та дані; базовий пріоритет; описувачі об'єктів; змінні оточення.

Задачі можуть перебувати в наступних станах [3].

READY. Задача запущена та готова прийняти на себе управління. Чекає тільки моменту коли на неї зверне увагу диспетчер. Коли диспетчер задач віддає їй процесорний час, то задача переходить в режим RUN.

RUN. Диспетчер перемкнув управління на задачу, процесор виконує безпосередньо її код в даний момент. У цей момент задача виконується.

WAIT. Задача знаходиться в стані очікування і чекає, коли відбудеться деяка подія, наприклад, поки пройде тайм-аут, або поки що-небудь в системі не станеться, на що ця задача повинна зреагувати. При цьому диспетчер не перемикається на неї, процесорний час не виділяється під цю задачу. Після того як очікувана подія відбудеться, то диспетчер задач призначить цієї задачі стан READY.

SUSPEND. Цей стан вимкнено. У цьому випадку задачу не вивантажено з пам'яті, дані її все збережені, але вона зараз неактивна. На жодні події не реагує і сама з цього стану вийде. Вивести її з цього стану можна тільки відповідною командою операційної системи вручну.

Також задачу можна видалити повністю. При цьому вона буде вивантажена з пам'яті, звільнить пам'ять, її поточний стан і локальні змінні будуть втрачені. Однак її можна буде при необхідності запустити знову.

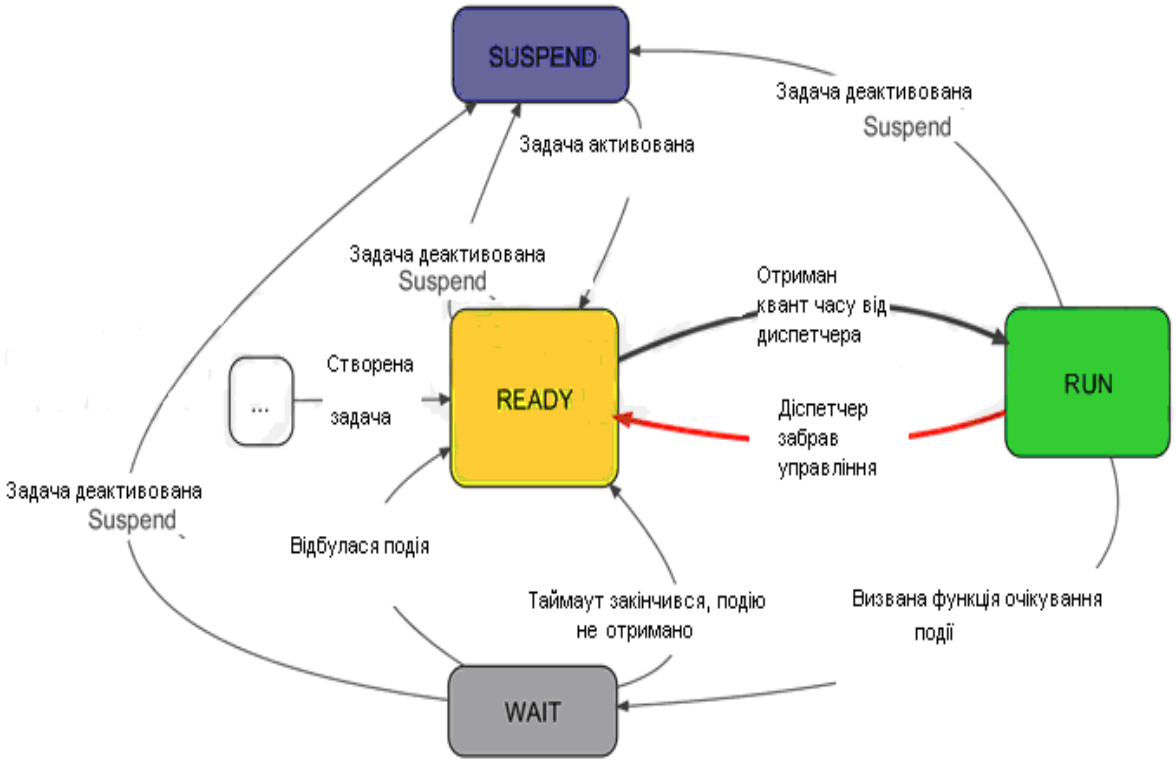

Рисунок 1.2 – Життєвий цикл задач

Кожній задачі при необхідності присвоюється пріоритет. Він задається при створенні задачі і його можна змінювати в процесі роботи. Саме від пріоритету планувальник визначає, в якому порядку будуть виконуватися задачі.

Якщо на даний момент часу запущених задач, які готові прийняти на себе управління немає, то диспетчер надасть права на роботу порожньому циклу IDLE. У цьому циклі відбувається обслуговування пам'яті, зачистка невикористаної оперативної пам'яті і так далі. Також в циклі IDLE можна мікроконтролер відправляти в режим енергозбереження.

Програмна реалізація задачі виглядає наступним чином:

```
void TaskN( void *pvParameters )
{
int variableN = 0;

for( ;; )
      {
      /* Код, який реалізує функціональність задачі */
```

```
        }
}
```

Тіло задачі реалізовано як нескінченний цикл. Змінні, які використовуються тільки функцією, оголошуються всередині функції. Також можуть бути і змінні типу static, тоді вони будуть доступні і інших функцій програми.

1.4 Планувальник

Управлінням станами задач займається планувальник задач або програма-монітор (sheduler). Програмна реалізація планувальника задач і є насправді реалізацією ядра ОСРЧ, тобто операційною системою. Від реалізації планувальника задач буде залежати правильність розподілу квантів часу і виконання всієї програми в цілому. При цьому в мікроконтролерах малого обсягу пам'яті планувальник може бути реалізований тільки у вигляді черги, в якій знаходяться потоки, чекаючи запуску на виконання [2].

Програма планувальника повинна забезпечувати взаємодію з усіма внутрішніми і зовнішніми пристроями, розподіляти пам'ять, визначати черговість виконання різних завдань, обробляти переривання і т.п. [5].

Після включення живлення планувальник повинен налаштувати мікроконтролер для виконання програми, що розробляється [3]. Для цього він має запрограмувати певні висновки мікросхеми мікроконтролера на введення або виведення інформації, включити і налаштувати внутрішні таймери і т. Д. Цей блок алгоритму програми-монітора називається ініціалізацією процесора. Ініціалізація мікроконтролера виконується тільки один раз, відразу після подачі напруги на мікросхему. Повторно ініціалізація може знадобитися тільки при збоях в роботі мікроконтролера. Загальний вигляд програми монітора наведено на рисунку 1.3.

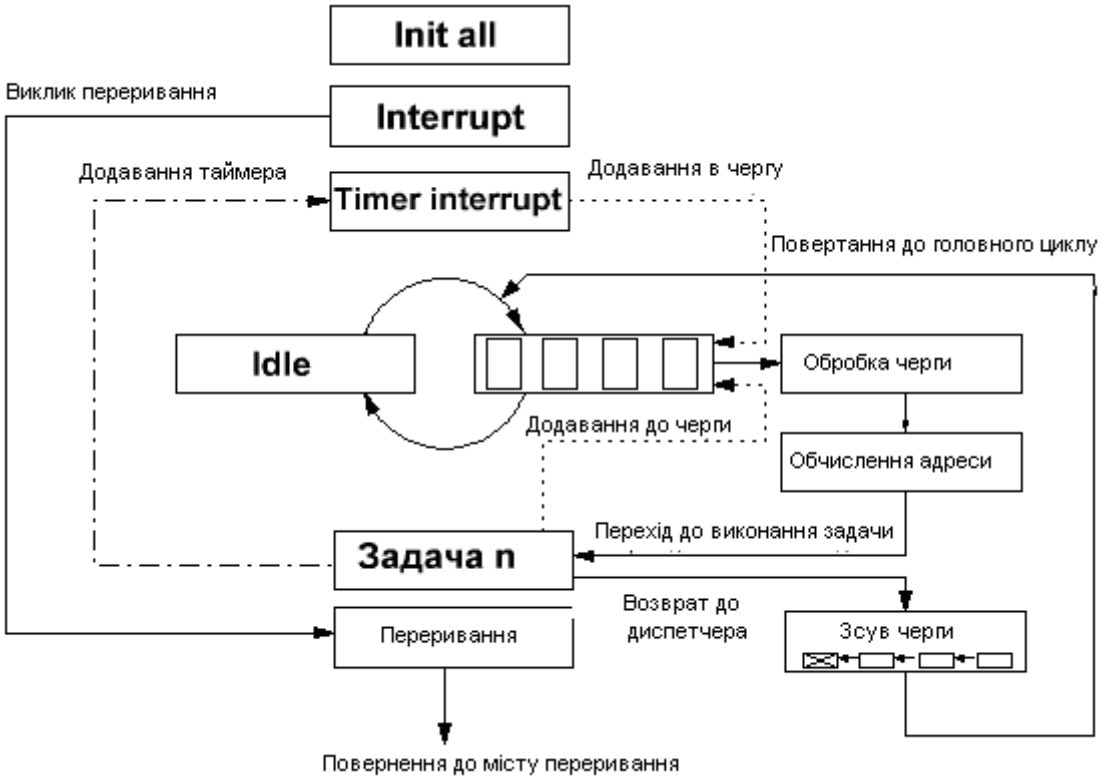

Рисунок 1.3 – Узагальнена діаграма роботи планувальника задач

Рисунок 1.4 – Схема алгоритму програми-монітора

Основна частина програми, що реалізує алгоритм роботи пристрою, починає виконуватися після ініціалізації мікроконтролера. Якщо в пристроях, що не містять програмно керованих компонентів, введення, обробка і виведення інформації виробляються різними апаратними блоками, то при виконанні програми ці ж дії проводяться послідовно одним і тим же пристроєм мікропроцесором. Для виконання кожної задачі зазвичай пишеться окрема підпрограма. Тобто при програмній реалізації пристрою підпрограма виконує ті ж функції, що і окремий блок при схемотехнічній реалізації пристрою.

Точно так, як і при апаратній реалізації різних блоків пристрою, необхідно, щоб кожна підпрограма вирішувала свою конкретну задачу. При написанні програми дуже часто виникає спокуса вирішувати проблеми в місці їх виникнення [7]. В результаті утворюється велика кількість ділянок коду, що займаються введенням інформації, ще стільки ж ділянок, що займаються управлінням одним і тим же пристроєм. Набагато краще, якщо введенням інформації займається одна підпрограма, для управління пристроєм, підключеним до мікроконтролеру, служить інша підпрограма, а загальний алгоритм роботи пристрою формує третя підпрограма. При цьому не можна допускати ситуації, коли підпрограма введення інформації тут же намагається обробити отримані дані, і тим більше почати управління будь-яким пристроєм.

Для того, щоб працювали всі написані підпрограми, вони включаються в один нескінченний цикл. Це еквівалентно періодичному запуску апаратних блоків. З'єднанню між блоками відповідає взаємодія частин програми, що здійснюється за допомогою глобальних змінних.

В цьому ж циклі зазвичай передбачається блок обробки помилок. Його призначення повідомляти оператору (користувачеві) про непередбачувану ситуацію, таку як неправильне введення з клавіатури або неправильні дані, отримані від підключеного до мікроконтролеру пристрою.

Програмна реалізація планувальника виглядає наступним чином:

```
{
  initTaskList(); // Створення черги задач
  runTasks(); // Виклик основного циклу з задачами
  while (1) {
    clearWatchDogTimer();
  }
}
```

Спочатку необхідно проініціалізувати всі пристрої, які використовуються в даному проекті, потім створити чергу задач, після чого викликається основний цикл із задачами, потім організовується нескінченний цикл. Також для захисту системи від збоїв бажано запустити WatchDogTimer, який захистить систему від небажаного зависання.

1.5 Взаимодія між задачами та розподіл ресурсів

Як розглядалося вище, задачі можуть надходити на вхід системи абсолютно незалежно. У зв'язку з цим виникає необхідність розподілу ресурсів в системі. Це може бути організовано двома способами. Перший спосіб FIFO («First In First Out») виконується наступним чином: нехай на вхід системи надійшла задача, вона отримує весь ресурс до закінчення виконання, після запускається наступна в черзі задача.

Другий спосіб називається каруселью багатозадачністю («round robin»). При ньому в системі весь час розбивається на кванти (time slice), які за певним алгоритмом розподіляються між задачами, які надходять в систему.

Є такі моделі диспетчеризації - кооперативна, витісняюча і інші.

Кооперативна багатозадачність – це тип багатозадачності, при якому операційна система одночасно завантажує в пам'ять два або більше додатків, але процесорний час надається тільки основним додатком, якщо на це отримано дозвіл основного потоку.

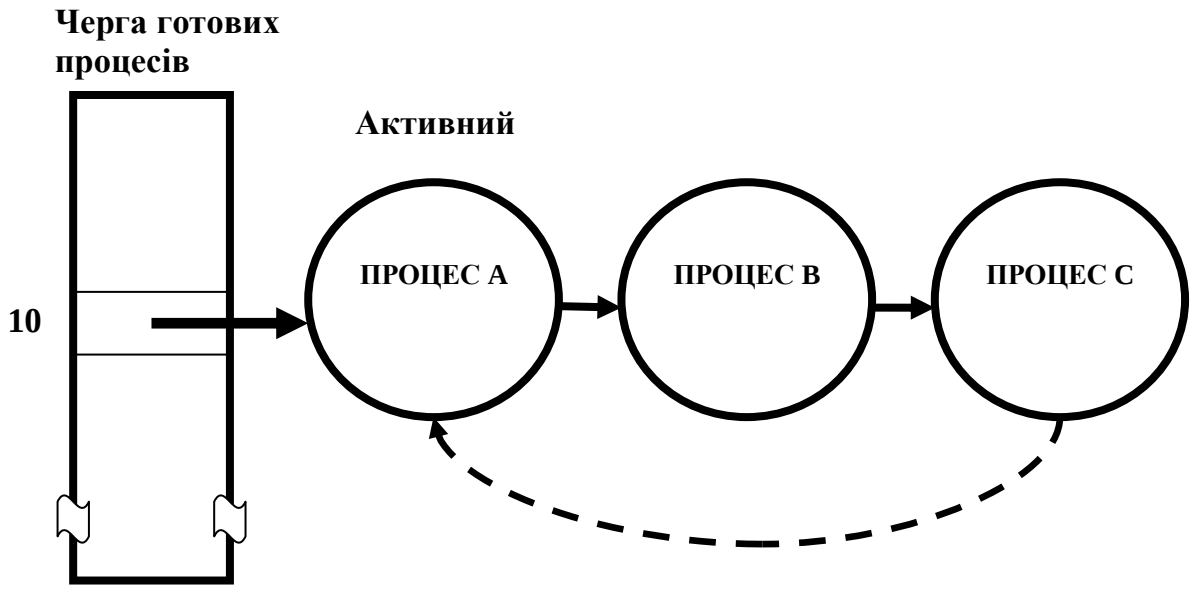

Рисунок 1.5 – Кооперативна багатозадачність

Кооперативну багатозадачність можна назвати багатозадачністю "другого ступеню" оскільки вона використовує більш передові методи, ніж просте перемикання задач, реалізоване багатьма відомими програмами. При простому перемиканні активна задача отримує весь процесорний час, а фонові задачі повністю заморожуються.

Якщо виконання основної задачі триває досить довго, то її виконання розбивається на невеликі ділянки, і після виконання такої ділянки процес добровільно поступається чергу іншим, а задача переходить в стан очікування.

При написанні планувальника по типу кооперативної багатозадачності необхідно дуже ретельно продумувати реалізацію при проектуванні кожного з паралельних процесів, так як інакше один процес здатний повністю монополізувати процесор, порушуючи роботу всіх інших процесів. У такому випадку деякі фонові задачі можуть взагалі не виконуватися через неправильний розподілу часового ресурсу.

Плюси кооперативної багатозадачності: простота реалізації, скромні вимоги до ресурсів пам'яті, сильне спрощення синхронізації взаємодіючих процесів.

При витісняючій багатозадачності процесорний час ділиться на кванти, і ці кванти виділяються процесам. Кожен процес має право використовувати свій квант, після чого планувальник перериває його виконання і надає черговий квант наступного готовому до виконання процесу, який ще своєї черги. Якщо процес не може використовувати свій квант повністю (наприклад, процес робить запит введення/виведення, який не може виконатися миттєво), цей процес тут же блокується до завершення операції, а квант передається наступному в черзі процесу.

Головне достоїнство витісняючого планувальника – гарантований розподіл процесорного часу. Навіть якщо один з процесів зациклився і перестав реагувати на адресовані йому події, це не призведе до краху системи в цілому: інші процеси як і раніше будуть отримувати і використовувати власні кванти процесорного часу.

Витісняюча багатозадачність має такі недоліки. Перший недолік – це зациклення процесу. Одна із задач високого пріоритету може захопити будь-який ресурс загального доступу на дуже великий проміжок часу, при цьому іншим доведеться чекати звільнення цього ресурсу. Це в свою чергу може порушити програму управління пристроєм.

Друга проблема, яка виникає при витісняючій багатозадачності, коли два процеси можуть розділяти між собою кванти часу, не залишаючи для інших задач можливість їх виконання. При цьому вийти з такого циклу самостійно їм навряд чи вдасться. Тому при розробці операційної системи слід приймати особливі заходи в синхронізації взаємодіючих потоків [9].

На рисунку 1.6 наведена схема роботи витісняючої багатозадачності для задач с різним пріоритетом, де 1 – переключення контексту, та запуск задачі Task_B; 2 – код задачі Task_B; 3 – переключення контексту та запуск задачі Task_A; 4 – код задачі Task_A; 5 – переключення контексту та

продовження виконання задачі Task_B; 6 – переключення контексту та продовження виконання фонової задачі.

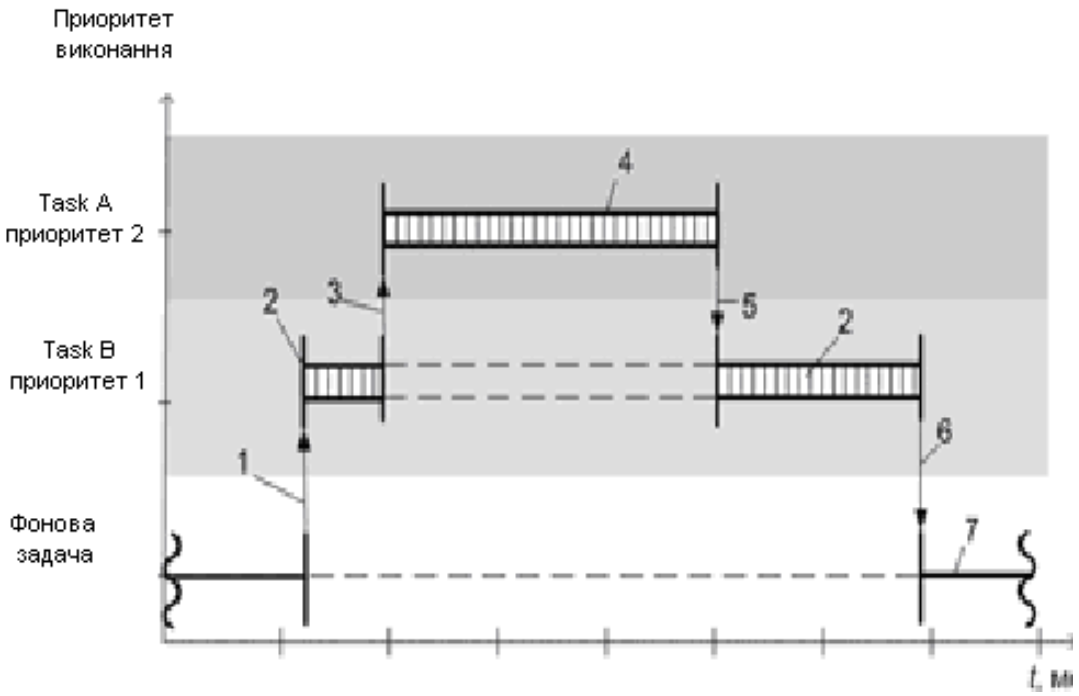

Рисунок 1.6 – Витісняюча багатозадачність з різними пріоритетами

На рисунку 1.7 наведена схема роботи витісняючої багатозадачності для задач з однаковими пріоритетами, де 1 – інтервал часу (Time Slice) завершено; 2 – код різних задач з однаковими пріоритетом; 3 – перемикання контексту, запуск наступної задачі; 4 – інтервал часу (квант).

Крім того, сам механізм витісняючої багатозадачності досить складний в реалізації та ресурсоємкий. Для мікроконтролерів малого обсягу пам'яті програм ця задача ускладнюється ще й тим, що для кожного з паралельних процесів потрібно зберігати потокові дані, наприклад стану регістрів процесора. В цьому випадку при виділенні пам'яті під контексти кожної задачі і таблиці планувальника обсягів пам'яті може не вистачити.

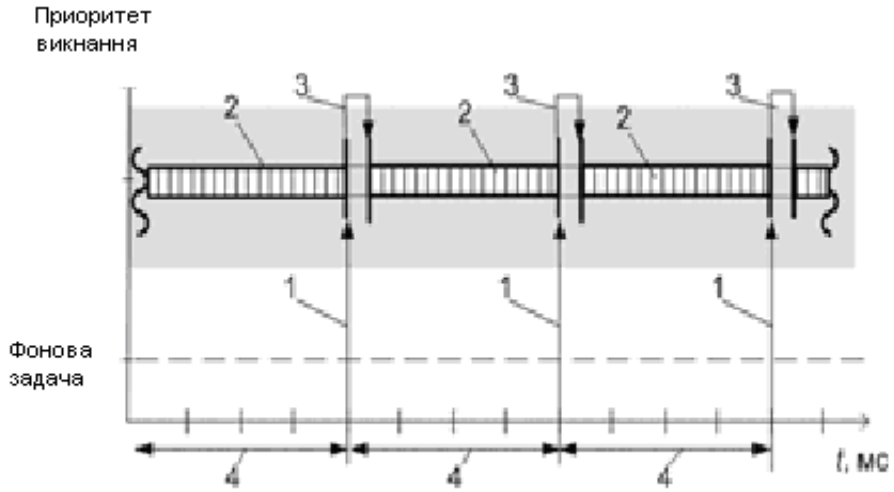

Рисунок 1.7 – Витісняюча багатозадачність з однаковими пріоритетами

1.6 Семафори та м’ютекси

Для упорядкування доступу до загальних ресурсів використовуються семафори і м’ютекси.

Семафори використовуються для синхронізації доступу до загальних ресурсів. Вони бувають бінарними або рахунковими. Якщо семафор бінарний, то він являє собою байтову змінну, яка може мати значення або «0», або «1». Задача, до якої підійшла черга виконуватися, захоплює семафор, інші задачі, яким потрібно використовувати загальний ресурс, чекають звільнення семафора. Лічильний семафор являє собою лічильник.

При доступі до загальних ресурсів також можуть бути критичні секції коду (глобальні змінні, зчитування даних і так далі). Для розрулювання таких ситуацій використовують механізм взаємовиключення доступу – м'ютексів. М'ютекс – це варіант семафора, який сигналізує іншим задачам, що ресурс зайнятий. Приклад програмної реалізації м’ютекса приведений нижче. У прикладі використовується глобальна змінна, яка інкременується в задачі.

```
int counter;
```

```
void task_N (void)
{
mutex_t mutex;

mutex_lock (& mutex);
global_counter++;

mutex_unlock (& mutex);
}
```

Якщо м'ютекс захоплений, то всі задачі, які використовують глобальну змінну, як наприклад в розглянутому вище прикладі це змінна counter, блокується. Після того як м'ютекс звільняється, всі інші задачі отримують доступ до ресурсу.

1.7 Обробники переривань

Крім чіткої черговості задачь, які виконуються за розглянутими вище правилами, в системі реального часу можуть бути використані зовнішні переривання від різних пристроїв. Але при цьому необхідно враховувати наступне.

По-перше, іноді виникає необхідність на заборону обробки будь-яких переривань, наприклад, в разі, коли втручання не допустимо (перетворення АЦП). У цьому випадку на початку виконання функції переривання забороняються, а потім заборона знімається. В іншому випадку це може викликати помилку. Тому, так як це і є критичні секції коду, то їх бажано уникати.

Необхідно, щоб такі частини коду були мінімально короткими. Їх періодичність також бажано знизити на скільки можливо. Для скорочення часу в обробник переривання допустимо виконувати тільки первинні дії, наприклад зчитувати первинних даних, а подальше обчислення вже виконувати в інших, не критичних до часу, частинах коду. При цьому обробник переривання виконує тільки самі «екстрені» дії, а основна обробка

«відкладується», поки її не виконає задача-обробник переривання. Така організація обробки переривань називається відкладеною обробкою. [7].

Після обробки переривання підпрограма обробника повертає управління планувальнику.

Після того, як обробка переривання буде завершена, можна виконувати перемикання задач, що і робить системна функція повернення з переривання, якщо виявляє, що прапор виставлений (саме для цього на початку потрібно зберегти контекст задачі) [5].

1.8 Огляд існуючих ОСРЧ

На сьогоднішній день існує більше 80 різних операційних систем реального часу, що підтримують 8-, 16- і 32-розрядні мікропроцесори [8-10]. Деякі з цих пакетів – закінчені операційні системи і включають не тільки ядро реального часу, але також менеджер введення-виведення, системи управління виводу інформації (дисплей), файлову систему, роботу з мережами, бібліотеки мов інтерфейсу, відладчики і міжплатформенні компілятори. У таблиці 1.1 представлені три операційні системи з підтримуваними мікроконтролерами.

Таблиця 1.1 – Операційні системи і підтримувані ними мікроконтролери

μC/OS	jacOS	Salvo
Analog Devices AD21xx	Microchip PIC12, PIC14, PIC16, PIC18	Atmel AVR, MegaAVR
Advanced Risc Machines ARM6, ARM7	Atmel AVR	Microchip PIC18
Hitachi 64180, H8/3xx, SH series	Texas Instruments MSP430	Texas Instruments MSP430
Intel 80x86, Pentium,	x51	

Pentium-II, 8051, 8052, MCS-251, 80196, 8096		
Mitsubishi M16, M32		
Motorola PowerPC, 68K, CPU32,		
CPU32+, 68HC11, 68HC16		
Philips XA		
Siemens 80C166, TriCore		
Texas instruments TMS320		
Zilog Z-80, Z-180		

Однак готові ОСРЧ необхідно адаптувати до конкретних програм і конкретному мікроконтролеру, що ускладнює іноді вирішення задачі проектувальника. Також готові операційні системи великовагові та вимагають певного обсягу пам'яті під виконання своїх завдань, що теж не завжди є оптимальним.

1.9 Особливості побудови систем реального часу

На основі проведеного аналізу сформулюємо вимоги які пред'являються до розроблюваної системі реального часу.

1 Вони повинні вбудовуватися в великі програмно-апаратні системи. При цьому взаємодія з іншими частинами такої системи повинні бути стандартизовані і розробляється система не повинна вносити істотні зміни в програмно-апаратну розробку при підключенні. Для цього в розроблювану систему повинні бути включені різні комунікаційні програми і драйвера.

2. Вони повинні здійснювати взаємодію з зовнішнім середовищем за допомогою опитування різних датчиків і виробляти керуючі впливу за допомогою різних механізмів або тривожних повідомлень і так далі.

3. ОСРЧ повинна бути побудована за принципом багатозадачності і при цьому допускати витіснення.

4. Обробка всіх вхідних впливів і вироблення управляючих сигналів повинні знаходитися в жорстких часових рамках, так як вихід за межі допустимих часових відхилень може привести до необоротних іноді наслідків. Тобто має бути чітко задано максимальний час.

5. Вона повинна будуватися на основі пріоритетів для різних задач, при цьому підтримувати передбачувані механізми синхронізації.

6. Повинні існувати механізми блокування та комунікації задач, які ділять між собою ресурси мікроконтролера.

2 РЕАЛІЗАЦІЯ АПАРАТНОЇ ЧАСТИНИ ПРОЕКТУ

2.1 Технічне завдання

Для реалізації системи управління рухомим об'єктом на основі операційної системи реального часу сформуємо технічне завдання з двома видами паралельно опитуваних датчиків.

Було сформовано наступне технічне завдання: рухомий об'єкт автономно повинен проходити через невідомий лабіринт, в якому розташовані міни. Рухомий об'єкт повинен самостійно знайти вихід з лабіринту при цьому об'їжджаючи міни.

Об'єкт повинен вибирати напрямок руху за допомогою інфрачервоних датчиків і заданої логіки руху, приймаючи рішення рухатися вперед або назад для продовження шляху, при цьому він повинен не нариватися на міни. Міни реалізовані за допомогою магнітів. Рухомий об'єкт повинен виявляти міни за допомогою датчика Холла. Якщо він знаходить міну, то він повинен від'їжджати назад і трохи змінювати напрямок руху для того, щоб об'їхати її.

2.2 Модель рухомого об'єкта

Для реалізації об’єкта, який буде здійснювати рух за заданими технічним завданням, необхідно розробити систему керування рухом.

Цей модуль повинен складатися з наступних компонентів:

– мікропроцесорної платформи, як модуля керування об’єктом;

– двигунів, які будуть здійснювати відповідно рух об’єкту;

– драйверу двигунів, який являє мікросхему, в яку інтегровані два моста для відсікання пікових викидів струму, що перевищують межу в момент запуску або зупинки двигунів;

– інфрачервоних датчиків, для знаходження стінок лабіринту;

– датчика Холла, який необхідний для виявлення мін.

2.3 Вибір апаратної платформи і сенсорів

Для здійснення вибору платформи, яка буде використовуватися як модуль управління, було зроблено огляд сімейства Arduino.

Сімейство Arduino включає в себе широкий ряд різних платформ, на яких встановлені різні мікроконтролери. Велика частина мікроконтролерів відноситься до сімейства AVR, але також є плата, реалізована на основі мікроконтролера ARM.

Плати Arduino прості у використанні і не мають вбудованих дисплеїв, кнопок (за винятком кнопки перезавантаження) або датчиків. Замість цього вони мають широку базу підключаються зовнішніх датчиків, дисплеїв, плат розширення та різних модулів зв'язку.

У результаті проведеного аналізу як модуль управління в проекті була використана платформа Arduino Uno як пристрій для керування двигунами та як модуль обробки даних від датчиків.

Плата Arduino Uno є базовою платою сімейства Arduino, має стандартну вихідну напругу - 5В і зручний діапазон напруги живлення від 7В до 12В. Arduino Uno має 6 аналогових виводів і 14 цифрових, що є достаточною кількістю для проектування різних пристроїв, а при необхідності в більшій кількості виводів до неї можна підключити плату розширення (shield). Співвідношення розмірів з кількістю виводів у даній плати найбільш зручно для роботи, тому що можна підключити досить велике число елементів і при цьому вмістити весь пристрій в невеликому корпусі.

Для реалізації проекту було обрано такі сенсори: інфрачервоний датчик перешкод (ІК), датчик Холла.

ІК датчик перешкод працює наступним чином: датчик випускає інфрачервоний сигнал зі світлодіода і потім ловить відображення цього інфрачервоного сигналу від перешкод фоторезистором. Він дозволяє

виявляти перешкоди на відстані від декількох сантиметрів до десятків сантиметрів. Датчик наведено на рисунку 2.1.

Рисунок 2.1 – Інфрачервоний сенсор перешкод

Дальність впевненого виявлення перешкод ІК датчика до 30 см і залежить від точної генерації випромінюваного сигналу, потужності випромінювання світлодіодів і особливостей поверхні виявленого перешкоди.

Для виявлення мін в лабіринті був обраний датчик Холла. Його принцип роботи побудований на зміні вихідної напруги в залежності від присутності або відсутності магнітного поля, в даному випадку мін – магнітів, що розташовані по різним місцям в лабіринті. При знаходженні магнітного поля міни датчик виявляє його та змінюється напруга струму вихідного сигналу.

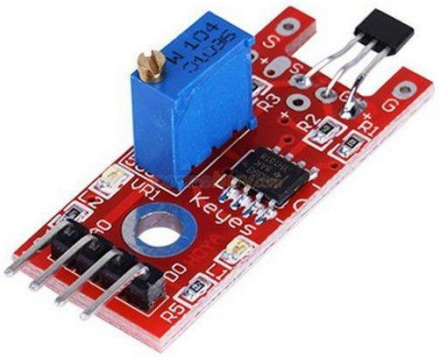

Рисунок 2.2 – Датчик Холла

Рух об’єкту буде здійснюватися за допомогою чотирьох двигунів. Плати Arduino, крім спеціалізованих, не підтримують можливостей керування двигунами постійного струму безпосередньо тому була використана схемна реалізація драйверу двигунів на основі мікросхеми L298N. Вона потужна, має хороший запас по максимальному струму (на канал). Схема підключення двигунів до драйверу двигунів та до платформи Arduino Uno наведена на рисунку 2.3.

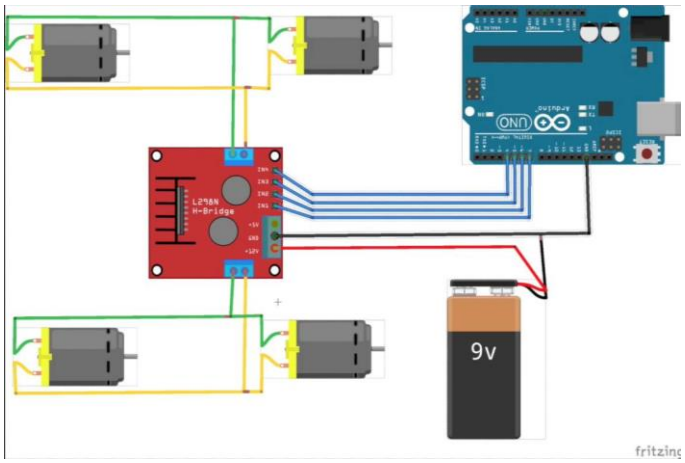

Рисунок 2.3 – Схема підключення двигунів до драйверу двигунів та до платформи Arduino Uno

2.4 Алгоритм руху об'єкта

На передній частині об'єкта встановлені три ІК датчика, які дозволяють виявити стіни лабіринту. На підставі даних від ІК датчиків мікроконтролер, приймає рішення про напрямок руху і формує сигнали для управління двигунами. Також на корпусі закріплений датчик Холла, що дозволяє виявляти магнітні «міни».

Об'єкт починає рух по лабіринту, при цьому весь час аналізуючи дані від датчиків. Якщо об'єкт наближається до стінки, то відповідний датчик визначає перешкоду і виробляється рішення про поворот в протилежну сторону, щоб уникнути зіткнення. Управління рухом включає дві складові:

логічний «0» – для руху назад, логічна «1» – для управління рухом вперед та задача управління швидкістю руху за допомогою широтно-імпульсної модуляції (ШИМ).

Якщо датчик Холла виявляє «міну», рухомий об'єкт від'їжджає назад і повертає вправо, після цього продовжує свій рух по лабіринту.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ ПРОЕКТУ

3.1 Функціональна схема програми управління

В роботі запропонована наступна функціональна схема операційної системи реального часу, для цього були визначені різні рівні її рішення (рис. 3.1).

Рисунок 3.1 – Функціональна схема реалізації багатозадачності

Механізм багатозадачності забезпечується за принципом поділу за кодом. Основна ідея полягає в розбитті коду задач на деякі невеликі частини, які виконуються безперервно. Тобто кожна з чотирьох задач системи повинна функціонувати по суті як кінцевий автомат.

На рівні додатку було запропоновано керуючу програму розділити на чотири паралельні задачі:

– управління двигунами;

– об'їзд міни;

– розрахунок оптимального руху;

– фонова задача, яка переводить мікроконтролер в режим сну для економії енергії в тому випадку, якщо в даний момент часу жодна задача не виконується.

На рівні прикладних програм було запропоновано розбиття на програму планувальника і програму обробки переривань. Планувальник підтримує чергу виконуваних завдань, він повинен бути реалізований у вигляді програмного автомата. Задача диспетчера – це програмне забезпечення, яке відповідає за фактичне перемикання контексту і забезпечення псевдопараллельності виконання задач в реальному масштабі часу.

Також запропоновано рішення обробки задач по перериванню у вигляді бінарного світлофора.

На апаратному рівні відбувається опитування всіх сенсорів та виконується управління двигунами рухомого об'єкту.

3.2 Розбиття на задачі

Для реалізації операційної системи реального часу, виходячи із запропонованого вище алгоритму руху і апаратної реалізації рухомого об'єкту, можна виділити наступні задачі

– задача управління двигунами (поворот вправо, вліво, вперед, назад) і розрахунок для ШИМ;

– задача опитування ІК датчиків;

– задача АЦП- перетворення даних від ІК датчиків;

– задача опитування датчика Холла;

– задача – розрахунок оптимального руху, тобто визначення повороту в залежності від сигналів від датчиків.

Проаналізуємо пріоритетність виконання задач. При проходженні лабіринту виникає велика ймовірність, що рухомий об'єкт може отримати сигнали від датчика Холла та від ІК датчиків одночасно. Система управління

не може обробляти ці два сигнали одночасно, так як в системі один процесор. При цьому обидві події є критичними, інакше рухомий об'єкт або підірветься на міні або вріжеться в стінку лабіринту. Отже, розглянутим задачам необхідно присвоювати відповідні пріоритети. Були запропоновані наступні пріоритети:

1 – задача виявлення міни, тобто обробка події від датчика Холла;

2 – задача руху назад при виявленні міни;

3 – задача читання ІК датчиків та АЦП-перетворення;

4 – задача: розрахунок оптимального руху.

3.3 Програмний модуль управління рухом

Для управління двигуном було об'явлено 6 цілочисельних глобальних змінних, які будуть відповідати за включення и відключення моторів, а також за швидкість обертання.

Листинг 3.1

```
#define dir1PinL  2    //Motor direction
#define dir2PinL  4    //Motor direction
#define speedPinL 6     // Needs to be a PWM pin to be able
              to control motor speed

#define dir1PinR  7    //Motor direction
#define dir2PinR  8   //Motor direction
#define speedPinR 5   // Needs to be a PWM pin to be able to
          control motor speed
  pinMode(speedPinL, OUTPUT); //left motor PWM pin
  pinMode(speedPinR, OUTPUT); //rignt motor PWM  pin
  pinMode(dir1PinL, OUTPUT); //left motor direction pin1
  pinMode(dir2PinL, OUTPUT); //left motor direction pin2
  pinMode(dir1PinR, OUTPUT); //right motor direction Pin 1
```

```
  pinMode(dir2PinR, OUTPUT); //right motor direction Pin 2
```

Було розроблено такі базові функції для керування рухом об’єкту у різних напрямах:

`Setup_motor_system` – для присвоєння глобальним змінним значень з номерами портів Arduino і перекладу використовуваних портів в режим виводу даних

`speedPinR` – для зручного керування швидкістю і потужністю з допомогою широтно-імпульсної модуляції.

Для керування двигунами були розроблені наступні функції: рух вперед, рух вправо, рух вліво, рух назад. Вони наведені у листингу 3.1.

Листинг 3.2

```
void go_Advance(void)  //Forward
{
  digitalWrite(dir1PinL, HIGH);
  digitalWrite(dir2PinL, LOW);
  digitalWrite(dir1PinR, HIGH);
  digitalWrite(dir2PinR, LOW);
}

void go_Left(void)  //Turn left
{
  digitalWrite(dir1PinL, HIGH);
  digitalWrite(dir2PinL, LOW);
  digitalWrite(dir1PinR, LOW);
  digitalWrite(dir2PinR, HIGH);
}

void go_Right(void)  //Turn right
{
  digitalWrite(dir1PinL, LOW);
  digitalWrite(dir2PinL, HIGH);
```

```
  digitalWrite(dir1PinR, HIGH);
  digitalWrite(dir2PinR, LOW);
}

void go_Back(void)  //Reverse
{
  digitalWrite(dir1PinL, LOW);
  digitalWrite(dir2PinL, HIGH);
  digitalWrite(dir1PinR, LOW);
  digitalWrite(dir2PinR, HIGH);
}

void stop_Stop()    //Stop
{
  digitalWrite(dir1PinL, LOW);
  digitalWrite(dir2PinL, LOW);
  digitalWrite(dir1PinR, LOW);
  digitalWrite(dir2PinR, LOW);
}
```

Для управління швидкістю обертання двигунів була розроблена функція `set_Motorspeed`, аргументи якої задають параметри швидкості.

Листинг 3.3

```
void set_Motorspeed(int speed_L, int speed_R)
{
  analogWrite(speedPinL, speed_L);
  analogWrite(speedPinR, speed_R);
}
```

3.4 Програмний модуль опитування датчиків

Для отримання даних з сенсорів були розроблені функції зчитування даних з датчиків.

У наступному листингу 3.3 наведені функція ініціалізації датчика Холла та функція опитування сенсора відповідно. Функція являє собою реалізацією задачі опитування датчика Холла для уникнення мін рухомим об’єктом.

Листинг 3.4

```
#define MFsensor  D2

void task1_read_MFsersor()
{
go_Back();    //якщо побачили міну то їдемо назад та
          повертаємо направо
  delay(10);
  go_Right();
}
```

Функція опитування датчика Холла виконується за перериванням. Для цього була створена наступна функція обробник переривання:

```
digitalWrite(MFsensor, 1);
  attachInterrupt(digitalPinToInterrupt(MFsensor),
read_MFsersor, RISING);
```

Функція ініціалізації та опитування ІК сенсорів наведена у листингу. Ця функція являє собою реалізацію задачі опитування ІК сенсорів та аналого-цифрового перетворення даних від сенсорів.

Так як ця задача не може бути коректно перерваною у зв’язку з тим, що АЦП перетворення не може бути припинено або відкладено на деякий час, в функції використовується мьютекс. В початку функції отримується мьютекс

за допомогою функції `mutexLock();`, яка блокує переривання даного потоку та виконується заборона на виконання будь-якого переривання. Коли АЦП перетворення виконано мьютекс більше не потрібен, заборону на переривання скасовується та викликається функція звільнення мьютекса `mutexUnlock();`.

Листинг 3.5

```
  pinMode(LFSensor_1, INPUT);
  pinMode(LFSensor_2, INPUT);
  pinMode(LFSensor_3, INPUT);
  Serial.begin(9600);

void task2_read_sensor_values()
{
  for (;;)
  {
    noInterrupts();
    mutexLock();
    sensor[1] = analogRead(LFSensor_1);
    sensor[2] = analogRead(LFSensor_2);
    sensor[3] = analogRead(LFSensor_3);
    interrupts();
    mutexUnlock();
  }
}
```

3.5 Програмний модуль управління рухом

Модуль управління рухом об'єкта будується відповідно технічного завдання. У цьому модулі аналізуються дані отримані від трьох ІК датчиків і

на підставі цих даних приймається рішення про напрямок руху. Алгоритм руху об'єкта наведено на рисунку 3.2.

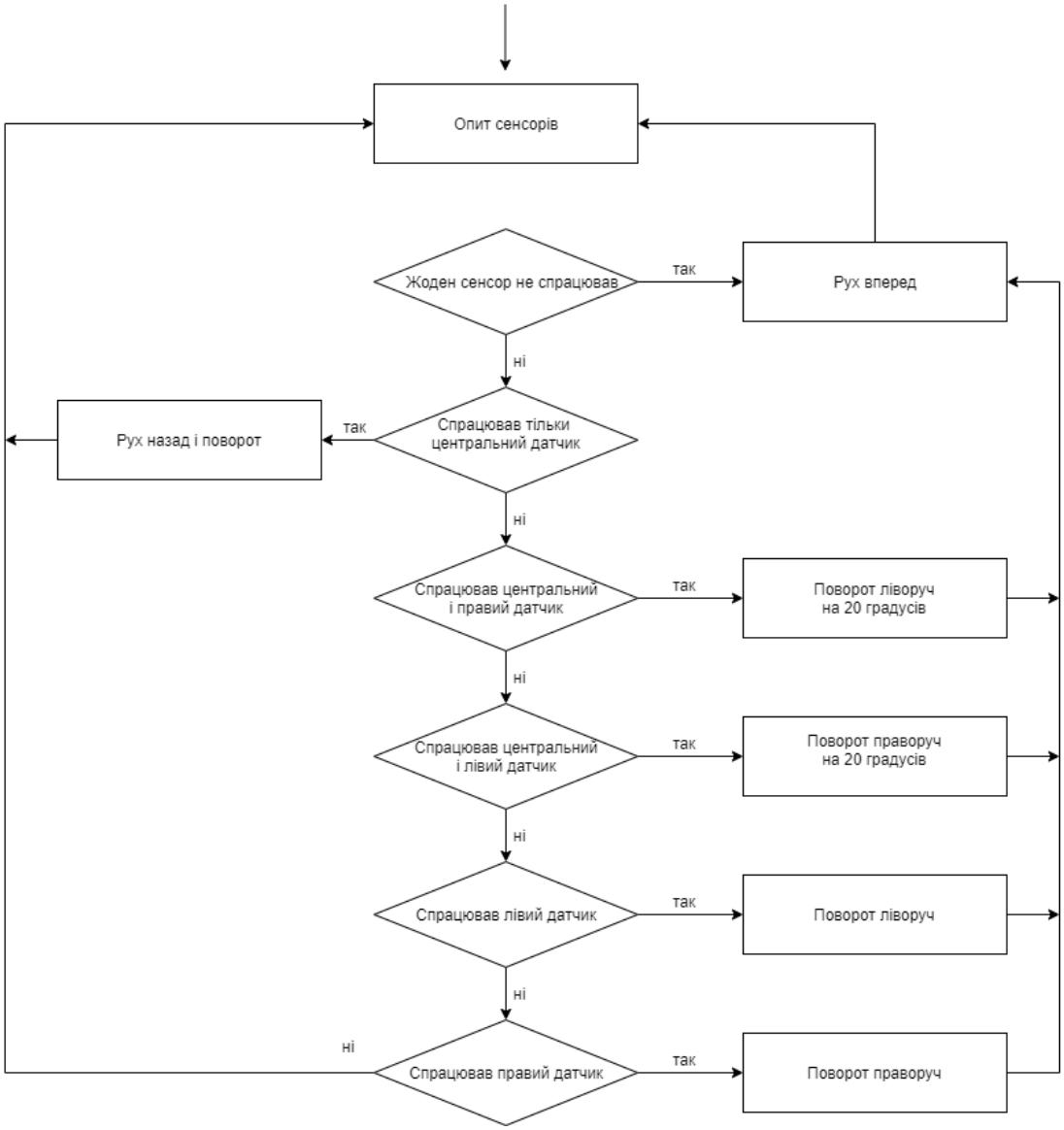

Рисунок 3.2 – Граф-схема алгоритму

На підставі граф-схеми алгоритму була розроблена модель керуючого автомату. Граф автомату наведений на рисунку 3.3.

Визначимо вхідні сигнали і вихідні сигнали записуються у вигляді x. Вхідні і вихідні сигнали виконуються при переході від одного стану до іншого, причому в такому порядку:

1. Вихідна дія стану, яке покинули.

2. Дія, що відбувається на ребрі переходу.

3. Вхідна дія нового стану.

Автомат виконує управління роботом в автономному режимі. При старті об’єкт рухається вперед до спрацювання якого-небудь з датчиків. Після знаходження будь якої перешкоди об’єкт виконує поворот у протилежний бік.

Перелік станів автомата наступний:

C0 – рух вперед;

C1 – поворот наліво на 20º;

C2 – поворот наліво на 90º;

C3 – рух назад;

C4 – поворот направо на 20º;

C5 – поворот направо на 90º.

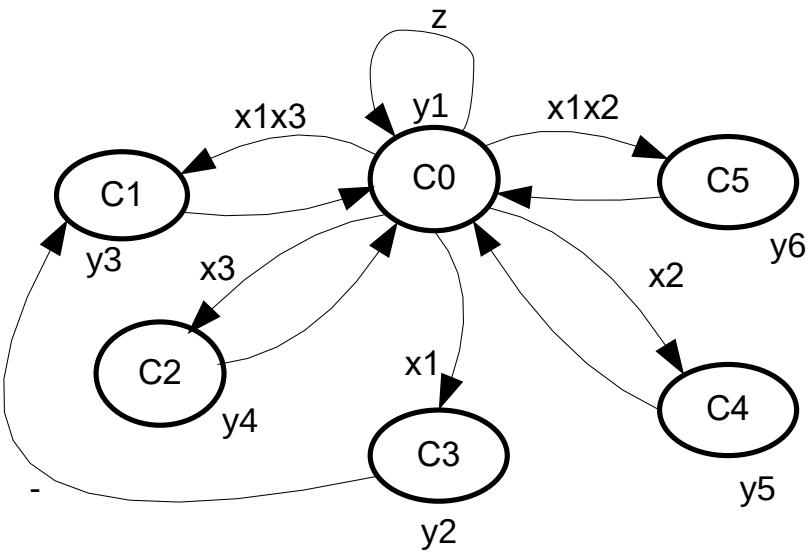

Рисунок 3.3 – Граф переходів автомата

Перелік вхідних сигналів автомата:

x1 – спрацював центральний датчик;

x2 – спрацював лівий датчик;

x3 – спрацював правій датчик;

z – сигнал опитування датчиків.

Перелік вихідних впливів автомата:

y1 – відправити команду «рух вперед»;

y2 – відправити команду «рух назад»;

y3 – відправити команду «поворот наліво на 20º»;

y4 – відправити команду «поворот наліво на 90º»;

y5 – відправити команду «поворот направо на 20º»;

y6 – відправити команду «поворот направо на 90º».

Реалізація програмного коду руху відповідно до запропонованого графу переходів автомату наведена у листингу

Листинг3.6

```
void task3_auto_tarcking()
{
```

```
      for (;;)
      {
        if (sensor[2] == opto_min_threshold) {
          if (sensor[1] == opto_threshold && sensor[3] ==
opto_threshold) {
            go_Advance();
            set_Motorspeed(M_SPEED1, M_SPEED1);
          }
          else if (sensor[1] == opto_max_threshold && sensor[3]
== opto_min_threshold) {
            go_Left();
            set_Motorspeed(0, M_SPEED1);
          }
          else if (sensor[1] == opto_min_threshold && sensor[3]
== opto_max_threshold) {
            go_Right();
            set_Motorspeed(M_SPEED1, 0);
          }
        }
        else if (sensor[2] == opto_threshold) {
          if (sensor[1] == opto_max_threshold && sensor[3] ==
opto_threshold) {
            go_Left();
            set_Motorspeed(0, M_SPEED1);
          }
          else if (sensor[1] == opto_threshold && sensor[3] ==
opto_max_threshold) {
            go_Right();
            set_Motorspeed(M_SPEED1, 0);
          }
        }
        else {
          go_Back();
          set_Motorspeed(M_SPEED1, M_SPEED1);
        }
```

```
        if (sensor[1] == opto_threshold) {
          if (sensor[0] == opto_max_threshold && sensor[2] ==
opto_min_threshold) {
            go_Left();
            set_Motorspeed(0, M_SPEED2);
          }
          else {
            go_Left();
            set_Motorspeed(0, M_SPEED1);
          }
        }
        if (sensor[3] == opto_threshold) {
          if (sensor[2] == opto_min_threshold && sensor[4] ==
opto_max_threshold) {
            go_Right();
            set_Motorspeed(M_SPEED2, 0);
          }
          else {
            go_Right();
            set_Motorspeed(M_SPEED1, 0);
          }
        }
      }
    }
```

3.6 Основний цикл та програмний модуль планувальника

Планувальник був реалізований як система з циклічним опитуванням. Це найбільш проста реалізація ядра планування ОСРЧ. Планувальник використовує механізм послідовного опитування, щоб визначити, чи потребує дана задача процесорного часу. Коли задача його вимагає, воно надається їй від початку до кінця виконання. Як тільки задача закінчується,

операційна система надає доступ до ресурсів задач, що вимагають процесорного часу.

Реалізація планувальника приведена в листингу

Листинг 3.7

```
taskSwitchinterrupt(uint32_t taskIndex) {
  Если стоит мьютекс, продолжаем выполнение текущей задачи
  if (getMutexStatus() == MUTEX_LOCK)
    return;

  pushCurrentTask(taskIndex);          // Сохроняем текущее
состояние стека задачи
  if (getTaskCnt() < taskIndex) { // Переход к следующей
задаче по кругу
    popNextTask(taskIndex + 1);
  } else {
    popNextTask(0);
  }
}
```

У листингу приведена основна функція. На початку створюється перелік задач (покажчики на функції). Потім відбувається ініціалізація таймера для відліку квантів часу. Створюється чергу задач та відбувається виклик основного циклу з задачами.

Листинг 3.8

```
void loop() {
  // список задач (показники на функції)
  tasks_pt      taskList[]      =      {&task1_read_MFsersor,
&task2_read_sensor_values, &task3_auto_tarcking};
```

```
        initTaskingTimer(10); // Таймер переключення задач кожні
10 мс
        initTaskList(taskList); // Створенні черги задач
        runTasks(); // Визов основного циклу з задачами

        while (1) {
          wait(100);
          clearWatchDogTimer();
        }
      }
```

4 ТЕСТУВАННЯ

В рамках проекту для тестування був зібраний наступний мобільний об’єкт, який побудований на базі платформи для створення роботів. Рамою для об’єкту виступають дві акрилові пластини, які не схильна до корозії, не проводить електрику, може використовуватися як в приміщенні так і на вулиці. Рама має отвори, які дозволяють встановити на об’єкт різноманітні датчики і плати.

На платформу було встановлено чотири двигуна постійного струму, блок живлення, плати управління та драйвер двигунів.

Інфрачервоні датчики відображення та датчик Холу були закріплені на передній частині об’єкту як показано на рисунку 4.1 та рисунок 4.2.

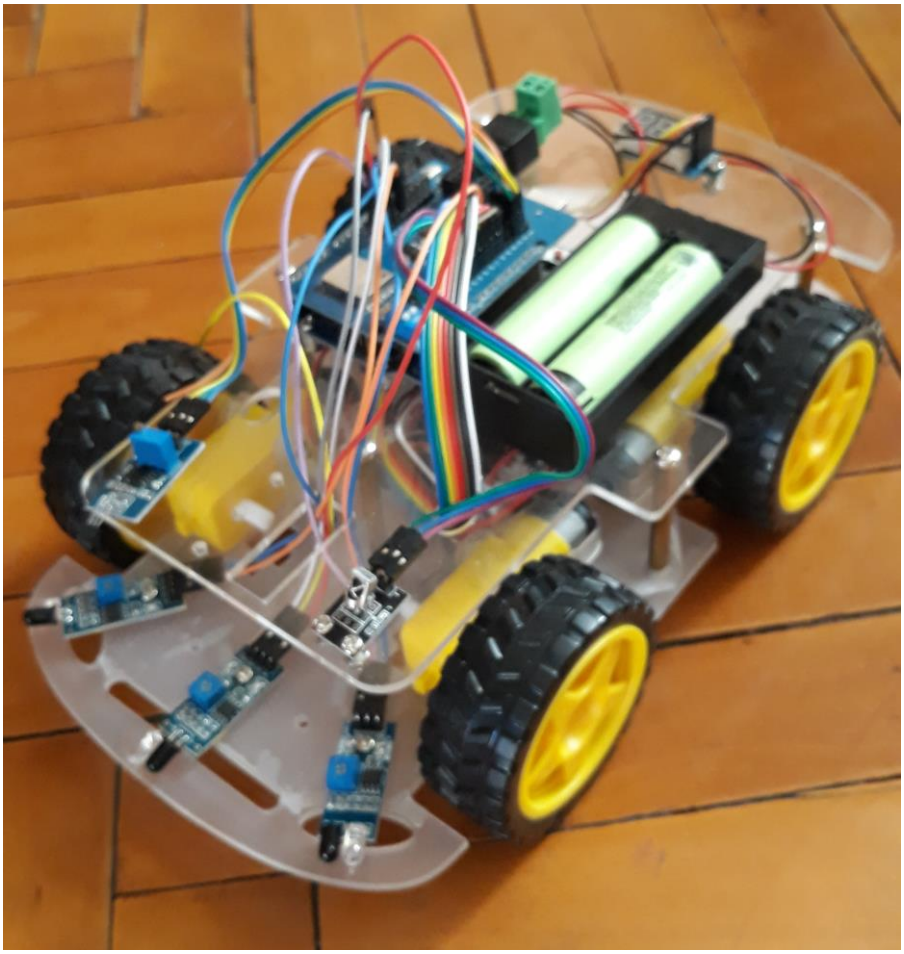

Рисунок 4.1 – Рухомий об’єкт

Рисунок 4.2 – Закріплення датчиків

Акумуляторна батарея була встановлена на пристрій та складається з літій-іонних акумуляторів типу 18650. Акумулятори мають високий показник по співвідношенню ємність/габарити. В батарейному блоці знаходяться два акумулятори, з'єднані послідовно, що забезпечує необхідну напругу на виході - 7-9В.

При тестуванні системи керування рухомим об’єктом враховувалися наступні показники:

– швидкість руху;

– точність руху, уникнення перешкод та повторюваність при проходженні лабіринту знову;

– гнучкість та надійність керування при різних умовах.

Тестування показало, що рухомий об’єкт проходив лабіринт достатньо точно. Об’єкт рухався без зупинок, обходячи всі перешкоди. При подальшому тестуванні змінювався лише час проходження лабіринту (в меншу та більшу сторони).

В результаті дослідження була розроблена операційна система. Вона показала стійку роботу на платформі Arduino. При цьому у розробленої системи є обмеження.

Було проведено тестування в спеціально виділеному ізольованому середовищі для безпечного виконання комп'ютерних програм Windows Sandbox. Для цього були підключені бібліотеки генератора випадкових чисел, був доданий емулятор зовнішніх подій, доданий віртуальний системний таймер і замість управління реальними виконавчими пристроями було здійснено текстовий висновок керуючих сигналів на консоль.

Результати моделювання наведені на рисунку 4.3, де можна бачити перемикання між задачами: опитування датчиків, вироблення керуючих сигналів для двигунів.

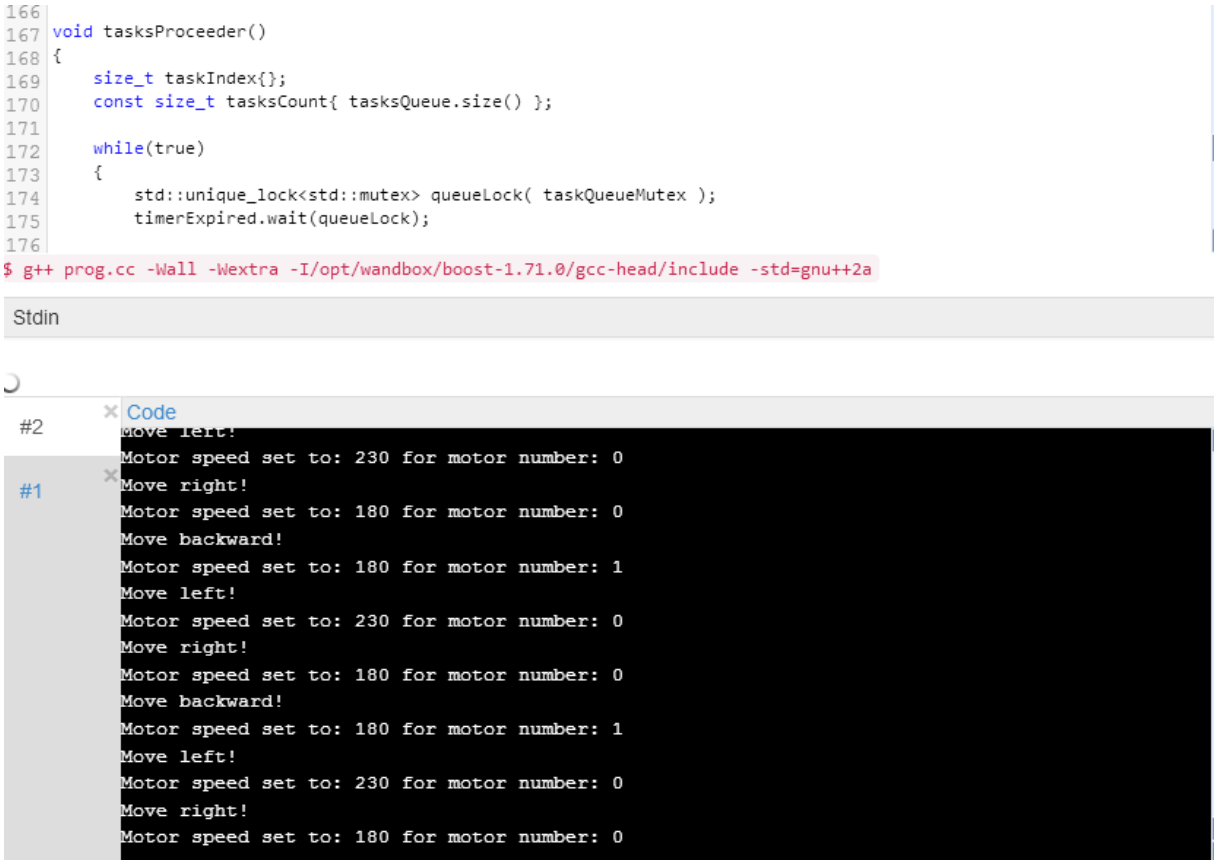

Рисунок 4.3 – Результати моделювання в середовищі Windows Sandbox

При компіляції та завантаженні порожнього скетчу з FreeRTOS на Arduino Uno можна побачити, що 20% флеш пам'яті споживається планувальником FreeRTOS, однак в розроблений в атестаційної роботі споживання пам’яті становить – 16%.

Розроблена операційна система володіє наступними функціональними можливостями:

– витісняючою та кооперативною багатозадачністю,

– відсутність обмежень на кількість задач, які може створити користувач;

– можливість періодичного запуску задач;

– використання двійкових або рахункових семафорів, м'ютексів.

Однак у запропонованій операційній системі не передбачена функція управління пам'яттю, функція статистики часу та виконання задач, відсутня черга повідомлень – для забезпечення міжзадачної комунікації.

Таким чином, хоча FreeRTOS і має більшу функціональність, але вимагає великих витрат оперативної пам'яті. Запропонована операційна система за умов обмеження таких ресурсів як час виконання, обсяг пам'яті є більш ефективною.

ВИСНОВКИ

В ході виконання атестаційної роботи були сформульовані вимоги, які пред'являються до систем реального часу. Були визначені такі поняття як задача, планувальник і сформульовані вимоги до них. Був проведений огляд існуючих операційних систем реального часу.

Було сформовано технічне завдання і на його підставі були розроблені апаратна і програмна моделі проекту. Була розроблена модель рухомого об’єкта, здійснено вибір апаратної платформи і необхідних сенсорів.

Також були розроблені функціональна схема керуючої системи. Реалізовано програмні модулі управління рухом, опитування датчиків і програмний модуль планувальника. Проведено тестування запропонованої системи на рухомому об'єкті.

В ході виконання атестаційної роботи було розроблено операційну систему реального часу, що надає формалізований набір засобів для розподілу завдань по організації потоку управління. Вона якісно змінює процес розробки програмного забезпечення в бік спрощення, формалізації логіки роботи програми як в межах одного процесу, так і в межах всієї програми в цілому.

Запропонована операційна система виконує наступні функції:

– перемикання між паралельними процесами;

– відлік таймаутів, витримку затримок;

– вибір готової до виконання задачі з найвищим пріоритетом і передача їй управління;

– обмін даними між задачами (за допомогою семафорів та м’ютексів).

Операційна система реального часу дозволяє програмісту зосередити свої зусилля на вирішенні конкретних завдань (алгоритмічних, математичних і т.п.), не відволікаючись на другорядні задачіі. Також розроблена операційна

система реального часу є більш ефективною ніж розглянуті за умов обмеження таких ресурсів як обсягу пам'яті, часу виконання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Борисов-Смирнов А. Операционные системы реального времени для микроконтроллеров [Электронный ресурс] / Электронный журнал №5 (129), 2008 Режим доступа: https://docplayer.ru/52758646-Operacionnye-sistemy-realnogo-vremeni-dlya-mikrokontrollerov.html– Загл. с экрана.

2 Смирнов, В.В. Метод организации параллельного выполнения задач в микроконтроллерах с малым объемом памяти программ / Н.В.Смирнова, В.В. Смирнов, Загальнодержавний міжвідомчий науково-технічний збірник Кіровоградського національного технічного університету / Конструювання, виробництво та експлуатація сільськогосподарських машин / вип. №28,

3 Микушин, А.В. Цифровые устройства и микроконтроллеры: учебное пособие/ А.В. Микушин, А.М. Сажнев, В.И. Сединин. – СПб.: БВХ-Петербург, 2010 – 832с.

4 Сулейманова, А.М. Системы реального времени: учебное пособие./ А.М.Сулейманова. – Уфа: Уфимск. гос. авиац. техн. ун-т, 2004. - 292 с.

5 Шалыто А. А. Автоматное программирование / Н.И. Поликарпова, А.А. Шалыто. – Спб.: Питер, 2008. – 167 с.

6 Шкіль, О.С. Мікропроцесорна система управління на основі багатозадачності / Шкіль О.С., Філіппенко І.В., Кулак Г.К., Семенцов Д.О. – // Інформаційно-керуючі системи на залізничному транспорті. – 2019. – № 4. – С. 75-76.

7 ГомаХ. UML. Проектирование систем реального времени, параллельных и распределенных приложений / Х.Гома. –М.: ДМК Пресс, 2011. - 704 с.:

8 Щербаков, К.С. Особенности работы операционной системы реального времени FreeRTOS // К.С. Щербаков, С.А. Щербаков. –Itech. Журнал интеллектуальных технологий. – 2011. – № 18. – С. 71–77.

9 Курниц, А. FreeRTOS — операционная система для микроконтроллеров / А. Курниц – Компоненты и технологии, № 3, 2011.

10 Бурдонов, И.Б. Операционные системы реального времени / И. Б. Бурдонов, А. С. Косачев, В. Н. Пономаренко – препринт Института системного программирования РАН, 2006 г.

11 National Instrument [Электронный ресурс]: What is a Real-Time Operating System (RTOS)? / Дата обращения: 24 ноября 2019. - Режим доступа: http://www.ni.com/white-paper/3938/en/

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

13 Уилмсхерст, Т. Разработка встроенных систем с помощью микроконтроллеров PIC. Принципы и практические примеры / Т. Уилмсхерст – Киев: МК_Пресс, СПб.: Корона_век, 2008, 544с.

14 Горошко, Е. Операционные системы реального времени / Е. Горошко – ИРЭ НАН Украины, 2003.

15 Рыжов. Д, [Электронный ресурс]: Разработка систем реального времени с использованием UML и каркасов приложений Режим доступа: http://www.myshared.ru/slide/111925/– Загл. с экрана.

16 Баранов, В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы / В.Н. Баранов. – М.: Додэка-XXI, 2004. – 288 с.

17 Голубцов М.С. Микроконтроллеры AVR: от простого к сложному / М.С.Голубцов. – М.: Солон-Пресс, 2003. – 288 с.

18 scmRTOS: Операционная система реального времени для однокристальных микроконтроллеров/ Руководство пользователя. – Новосибирск, 2006. – 109 с.

19. Timmerman, M. RTOS Evaluation Kick Off! // Martin Timmerman, Bart Van Beneden, Lourent Uhres – Real-Time Magazine. – 1998. – N3 pp 6 – 10

20 Сорокин., С. Системы Реального Времени. / С. Сорокин. – СТА. – 1997. - №2. – С 22-29.

21 Верхалст, Э. Задача разработки ОСРВ для цифровой обработки сигналов/ Э. Верхалст.– Мир компьютерной автоматизации. – 2002. – №4.

22 Zalewski, J.. What Every Engineer Needs To Know About Rate-Monotonic Scheduling: A Tutorial// Real-Time Magazine. – 2005. – N 1. – P 6-24.

23 Keeling, N.J.. Missed it! - How Priority Inversion messes up real-time performance and how the Priority Ceiling Protocol puts it right// Real-Time Magazine. – 1999. – N4. – P. 46-50.

24 Блискавицкий, А. [Электронный ресурс]: Операционные системы реального времени/ А. А. Блискавицкий, С. В. Кабаев. – Средства и системы компьютерной автоматизации. Режим доступа: http://www.asutp.ru.– Загл. с экрана.

25 Dedicated Systems Experts, QNX® RTOS 6.1// Dedicated Systems (www.dedicated-systems.com) . – 2002. – 97 p.

26 Parker, L. E. Path Planning and Motion Coordination in Multiple Mobile Robot Teams. Encyclopedia of Complexity and System Science. R. A. Meyers (ed.), Springer Verlag, 2009, pp.

27 Шалыто, А.А. Алгоритмизация и программирование для систем логического управления и «реактивных» систем. / А.А. Шалыто .– «Автоматика и телемеханика», 2001г., №1, с.3-39.

28 Шалыто, А.А. Switch-технология – автоматный подход к созданию программного обеспечения «реактивных» систем / А.А. Шалыто, Н.И. Туккель. – «Программирование», 2001г., №5, с.45-62.

29 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.

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

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