ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут Інформаційних технологій
Кафедра Інженерія програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр
на тему: «**Багатозадачність на Ардуїно на основі операційної системи реального часу FreeRTOS**»

Виконав: студент 4 курсу, групи ПД–42
спеціальності
121 - Інженерія програмного забезпечення
(шифр і назва спеціальності)
Карпенко Б.А
(прізвище та ініціали)
Керівник Корецька В.О.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут Інформаційних технологій

Кафедра _Інженерія програмного забезпечення _____
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки -121 –«Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерія програмного забезпечення
_____О.В. Негоденко
____ _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

КАРПЕНКО БОГДАНУ АНАТОЛІЙОВИЧУ
(прізвище, ім‘я, по батькові)

1.Тема роботи: «**Багатозадачність на Ардуїно на основі операційної системи реального часу FreeRTOS»**

Керівник роботи: Корецька Вікторія Олександрівна, доцент кафедри, кандидат педагогічних наук, доцент_____
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «18» лютого 2022 року №_____.

2. Строк подання студентом роботи 3 червня 2022 року

3. Вихідні дані до роботи:

3.1. Технічні характеристики мікроконтролерів Arduino.

3.2. Операційні системи реального часу.

3.3. Науково-технічна література з питань, пов‘язаних з системами управління, мікроконтролерами та операційними системами реального часу

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Операційна систем реального часу.

4.2. Модель рухомого об’єкту

4.3. Апаратна частина.

4.4. Програма управління

5. Перелік графічного матеріалу (12 слайдів)

5.1. Титульний слайд.

5.2 Мета, об’єкт, предмет, наукова новизна дослідження.

5.3 Актуальність.

5.4 Аналіз аналогів.
5.5 Технічні завдання.
5.6 Програмні засоби та інструменти реалізації.

_____

6. Дата видачі завдання «11» квітня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.2022	Виконано
2	Аналіз останніх публікацій і дослідженя	13.04.2022	Виконано
3	Вибір і аналіз засобів реалізації проекту	14.04.2022	Виконано
4	Проектування та реалізація	02.05.2022	Виконано
5	Вступ, висновки, реферат	17.05.2022	Виконано
6	Розробка обов'язкових демонстраційних креслень	18.05.2022	Виконано
7	Попередній захист роботи		
8	Подання роботи в деканат	06.06.2022	

Студент _____ Карпенко Б.А.

(підпис) (прізвище та ініціали)

Керівник роботи_____ Корецька В.О.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 59 с., 1 табл., 13 рис., 15 джерел.

Ключові слова: БАГАТОЗАДАЧНІСТЬ, ОПЕРАЦІЙНА СИСТЕМА РЕАЛЬНОГО ЧАСУ, ARDUINO

Об’єкт дослідження – процес побудови складних багатозадачних систем.

Предмет дослідження – дослідження операційних систем реального часу для організації багатозадачності на Arduino

Мета роботи – покращення організації багатозадачності на Arduino за допомогою розробленої програми на основі операційної системи реального часу FreeRTOS.

Наукова новизна – Створення програми з багатозадачністю на базі операційної системи реального часу FreeRTOS.

У дипломному проекті з використанням ОСРЧ FreeRTOS було розроблено програму управління рухом об’єкту в лабіринті з уникненням зіткнення з перешкодами завдяки ІЧ датчиків та датчика Холла.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 9

ВСТУП 10

РОЗДІЛ 1. ОПЕРАЦІЙНА СИСТЕМА РЕАЛЬНОГО ЧАСУ 11

1.1 Архітектура та поняття операційної система 11

1.2 Робота системи в реальному масштабі часу 13

1.3 Задачі та їх пріорітети 14

1.4 Планувальник задач 17

1.5 Взаимодія задач та розподіл ресурсів 21

1.6 Семафори та м’ютекси 25

1.7 Обробники переривань 26

1.8 Огляд існуючих ОСРЧ 27

1.9 Особливості побудови системи в реальному часі 28

РОЗДІЛ 2. ОГЛЯД ТА РЕАЛІЗАЦІЯ АПАРАТНОІ ЧАСТИНИ 29

2.1 Платформа Arduino 29

2.2 Техничне завдання для Апаратної частини 29

2.3 Модель Об’кта 30

2.4 Вибір апаратної платформи і сенсорів 30

2.5 Алгоритм руху об'єкта 33

РОЗДІЛ 3. ОГЛЯД ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ 34

3.1 Інтегроване середовище Arduino IDE 34

3.2 Мова програмування 37

3.3 Технічне завдання для програмної частини 37
3.4 Розподілення на задачі програмної частини 39
3.5 Програмний модуль управління двигуном 40
3.6 Програма моніторингу сигналів датчиків 43
3.7 Програмна реалізації керування рухом 45
3.8 Програмний модуль Планувальник 50
РОЗДІЛ 4. ТЕСТУВАННЯ ПЗ 52
ВИСНОВКИ 54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

КА – керуючий автомат;

МК – мікроконтролер;

САПР – система автоматизованого проектування;

ОСРЧ – операційна система реального часу;

ІК – інфрачервоний датчик;

ШІМ – широтно-імпульсна модуляція.

АЦП – аналого-цифровий перетворювач

ВСТУП

З початку появи мікроконтролерів та їх стрімкого розвитку у всіх галузях повсякденного життя вони використовуються для упровляючих завдань в різних пристроях. Вони призначенні для управління системами, згідно з розробленною мікроконтролерною програмою. Їх використовують, як в побутових системах, так і наприклад у промислових. З кожним роком кількість завдань котрі повинні виконуватися мікроконтролером паралельно, постійно збільшується. З цього виходить, що мікроконтролерні програми стають більш складними, як в написанні, так і в налагоджені. При розробці будь-якого складного програмного забезпечення для мікроконтролерів стає зрозуміло, що програма буде складатися з декількох, відносно самостійних завдань з комунікаціями задач між собою.

З цього маємо що є потреба в керуючій програмі що повинна працювати в реальному масштабі часу. В основі розвитку зараз є 8-ми та 32-х розрядні мікроконтролери. На ринку багато операційних систем реального часу які з повнотою можуть покривати потреби багатозадачності в мікроконтролерах. Однією особливостей є велика різниця між написанням программ для мікроконтроллерів та універсальних комп'ютерів. Операційна система виконує програми такі як запуск та інші, також взаємодіє із зовнішніми та внутрішніми пристроями або цим займається людина. У мікроконтролері це виконує програма.

Об'єктом дослідження є процес побудови складних багатозадачних систем.

Предметом дослідження – дослідження операційних систем реального часу для організації багатозадачності на Arduino

Метою роботи є покращення організації багатозадачності на Arduino за допомогою розробленої програми на основі операційної системи реального часу FreeRTOS.

Наукова новизна проекту – створення програми з багатозадачністю на базі операційної системи реального часу FreeRTOS.

РОЗДІЛ 1. ОПЕРАЦІЙНА СИСТЕМА РЕАЛЬНОГО ЧАСУ

1.1 Архітектура та поняття операційної система

При розробці програмного забезпечення на мікроконтролерах постає задача розподілу часу між задачами, майже кожна вбудована система має більш ніж одну задачу для обробки в реальному масштабі часу. З цього маємо що система повинна процювати при чотких часових обмеженнях.

Програма управління об’єктом чи об’єктами, може включати в себе багато функцій які меж собою не як не пов'язані, наприклад як освітлення так і визначення відстані до об’єкту.

В залежності від поставлених завдань та умов їх виконання таких як опитування різних датчиків, управління освітленням, оповіщення та інше, між їх виконанням доводиться вирішувати та програма управління все більш ускладняється. Складність балансування послідовності дій котрі система повинна виконувати та визначення черги виконання послідовності без шкоди для інших задач. Кожна задача буде заважати та конкурувати за процесорний час, це може викликати затримки в обробці інших задач системи, за рахунок чого якість виконання буде менша. Тобто багатозадачність не буде виконана із за виконання лише одного завдання чи некоректного виконання завдань.

Таким чином, система повинна мати безліч незалежних вхідних потоків даних від джерел та керувати їми.

За не можливості передбачення часу надходження даних з кожного датчика , все ж необхідно реагувати на подію не порушуючи часові обмеження, для того щоб часові обмеження сформульовані в технічному завданні не були не дотримані.

За рахунок збільшення кількості датчиків, з’являється вірогідність появи даних в один час з декількох джерел, також збільшується ймовірність колізії

сигналів, що надходять та які потребують обробки. Через це код програми починає ускладнюватися. Та за рахунок цього одна задача програми починає відповідати за виконання декількох задач одночасно. Переривання починають передоватися по одному каналу звязку що зменшує швидкість, за рахунок додаткових апаратних блоків. Також проблеми виникають так як: обмеження обсягу памяті мікроконтролера, чіткий регламент обробки переривань та критичних ділянок программ, відсутність компактної ОС.

Усі ці питання можливо вирішити завдяки операційній системі реального часу.

Операційна система реального часу скорочено ОСРЧ дозволяє забезпечити багатозадачність систем завдяки сервісів, котрі надаються ядром, що дозволяє по перше раціонально використовувати ресурси процесора та по друге спростити розробку програмного забезпечення та збільшити ефективність.

Система реального часу яка обробляє інформацію з декількох датчиків наведено на рисунку 1.1.

Спільне використання ресурсів мікропроцесорів є важливим паралельним аспкектом для необхідності того, щоб все відбувалось завдяки реальному маштабу часу. Вбудовані системи мають проблеми конкуруючих програм, які виріщуються за допомогою керуючої програми, за рухунок яких складніше розпредиляти процесорний час.

Забезпечення передбачуваності чи детермінованості системи при яких завгодно критичних впливах на систему, це є основна вимога до операційної системи реального часу. Це і відрізняє вимоги ОСРЧ від продуктивності та швидкодії стандартних ОС.

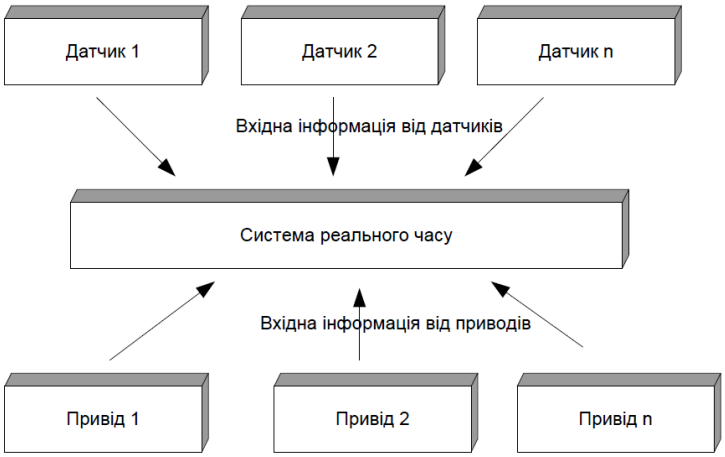

Рис 1.1 – Система реального часу

1.2Робота системи в реальному масштабі часу

Розглянемо сформульоване Гілліосем Дональдом 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».

Визначення що було указно вишче можливо трактувати інакше, СРЧ знаходиться у жорстких чавоих рамках. Часові рамки вариативні, велики та маленьки, тобто обробка інформації швидка. Є два варіанти: реакція системи та обробка сигналів що надходять задовільна до техничного завдання, то система відповідає вимогам. Якщо порушено часові рамки то вважається що у системі непропустимий збій.

З цього маємо, що швидко дія системи системи необхідна для швидкого протікання поверхневих подій для котрих СРЧ має відстежувати та виробляти упровляючи дії. Що можна рахувати як чемність роботи системи.

ОСРЧ діляться на класи такі як: “Жорстка”, “М`яка” за часовими обмеженнями.

М’яка система має запас часу при реакції на систему. Від жорсткої відрізняється насамперед затримкою яка може бути отримана з невеликим запізненням за рахунок цього не відбудеться збій, але відбудеться зниження продуктивності системи. Також немає критерію для проміжка часу за котрим можливо затягувати системну відповідь на сигнал.

Головні відмінності між класами систем реального часу такі: жорстка система не може запізнюватися за реакцією на подію, м’яка система реального часу має таку можливість на запізнення з реакцією на подію.

1.3. Задачі та їх пріорітети

Під завданням розумітимемо послідовність операцій, які вирішують алгоритм. Інакше кажучи, цей фрагмент коду, з погляду програміста, має виконуватися «одночасно» коїться з іншими завданнями і може працювати незалежно від інших фрагментів коду.

Завдання можна виконувати безпосередньо в суворій послідовності. Завдання у цьому випадку виконується одне за одним, з прервами, потім програма вдовольняє цім умовам завдяки циклам, умовним переходам, викликами підпрограм. Кожне завдання може мати пріоритети. Призначення пріорітетів між завданнями здійснюється в кожній задачі залежно від важливості технічного завдання та умов часу.

Такі поняття як “задача” та “потік” у мікроконтролерах з малою програмною пам’яттю потрібно вважати синонімами.

Коли так відбувається то необхідно до кожного потоку вказати наступне: адресний простір, виконавчий код, дані, базований пріоритет, змінне оточення, описувач об’єктів.

Існує декілько станів задачі.

READY. Завдання працює і готове взяти під контроль. Чекають моменту, коли диспетчер його помітить. Коли диспетчер передачі закінчується на центральному блоці обробки, робота переходить у режим RUN.

RUN. Диспетчер перемкнув управління на завдання, і процесор зараз виконує код безпосередньо. Наразі завдання працює.

WAIT. Завдання знаходиться в режимі очікування і очікує настання будь-якої події, наприклад, до закінчення терміну або до того, як щось станеться в системі. При цьому диспетчер не може перейти до нього, а процесорний час не виділяється на це завдання. Як тільки відбудеться очікувана подія, диспетчер завдань визначить стан, готовий до роботи.

SUSPEND. Ця умова вимкнена. У цьому випадку завдання не видаляється з пам’яті, всі його дані зберігаються, але на даний момент воно не активне. Він не буде реагувати ні на один з інцидентів і самостійно вибереться з цієї ситуації. Його можна вивести з цього стану лише відповідною командою операційної системи.

Також можете виконати завдання повністю. У цьому випадку вона буде вивонтажена з пам’ять, звільнить пам’ять, а її поточний стан і локальні змінні зникнуть. Однак при необхідності його можна перезапустити.

Рис1.2. Життєвий цикл

Кожному завданню надається пріоритет, коли це необхідно. Він встановлюється під час обробки проблеми і може бути змінений в процесі роботи. Планувальник визначає порядок, у якому будуть виконуватися завдання.

Якщо в даний момент роботи немає, диспетчеру буде дозволено працювати у вільному циклі “Бездіяльність системи”. У цьому циклі обслуговується пам’ять, очищається невикористана оперативна пам’ять тощо. Також в циклі “Бездіяльність системи” можливо перевести мікроконтролер в режим енергозбереження.

Програмне забезпечення завдання таке:

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

}

Тіло задачі реалізоване нескінченним циклом. Використовуємі змінні у функції потрібно оголошувати всередені. Также є змінні статичного типу, вони включаються в інші функій.

1.4. Планувальник задач

Планувальник управляє станами задач. Реалізація планувальника цє і є реалізація ядра ОСРЧ.

Від реалізації планувальника задач залежить правильний розподіл часу та загальна продуктивність програми. У той же час у мікроконтролерах з невеликим об’ємом пам’яті планувальник може бути реалізований лише у вигляді черг, де очікується виконання потоків.

Програма-планувальник забезпечує взаємозв’язок з усіма внутрішніми та зовнішніми пристроями, виділяє пам’ять, визначає порядок виконання різноманітних завдань, обробляє переривання тощо. Після того як буде подане живлення, планувальник починає налаштовувати мікроконтролер для виконання програми шо була разробленна. Деякі мікросхеми за для цього будуть запрограмовані. Включити та налаштувати внутрішні таймери за для введення та/або виведення даних. Цей блок планувальника являє собою ініціалізацією процесора. Ініціалізація відбувається один раз тоді коли відбувається подача напруги. Якщо мікроконтролер не працює, можливо необхідне повторна ініціалізація. Загальний огляд планувальника задач рисунок 1.3.

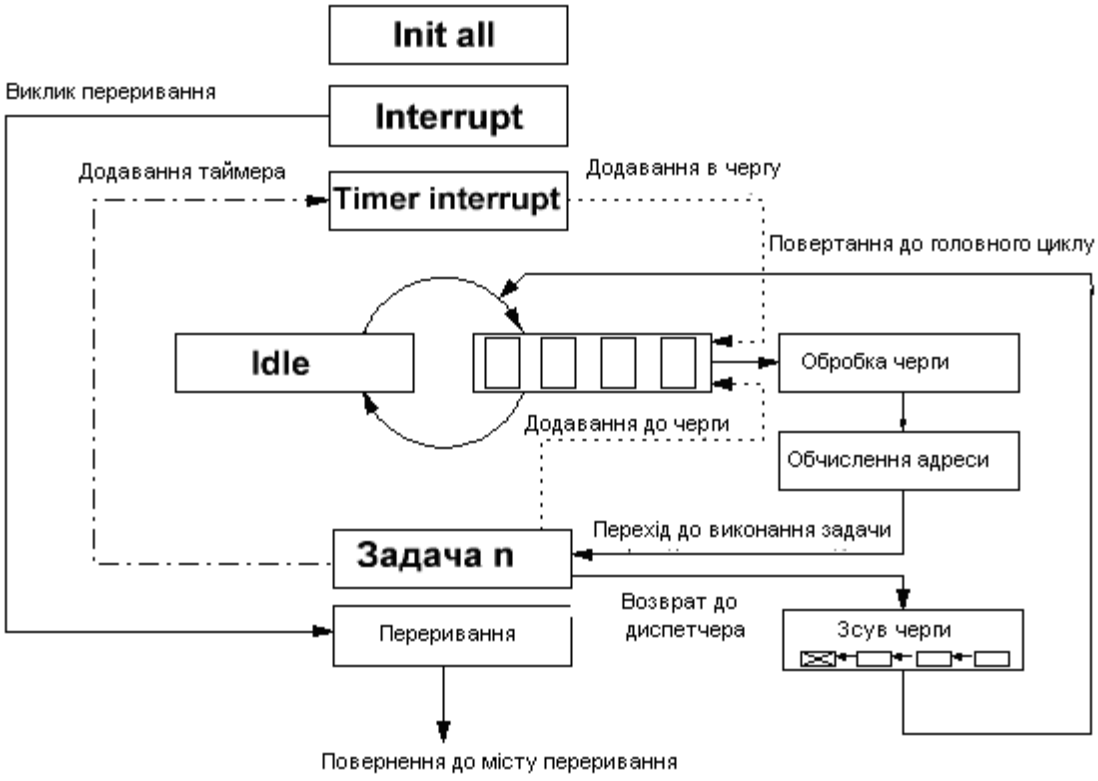

Рис1.3. Узагальнена діаграма роботи планувальника задач

Рис1.4. Схема алгоритму програми-монітора

Після запуску мікроконтролера, починає виконуватися головна частина програми, яка реалізує алгоритм пристрою.

Якщо введення, обробка та виведення інформації на пристроях без програмно-керованих компонентів здійснюються різноманітними апаратними блоками, то одні й ті ж самі дії послідовно виконуються ті ж самим мікропроцесорним пристроєм під час виконання програми. Для виконання кожного задання здебільшого потрібно написати окрему підпрограму. Іншими словами при програмної реалізації пристрою підпрограма як і окремий блок виконують однакові функції при схемотехнічній реалізації.

Так само, як і апаратна реалізація різноманітних блоків пристрою, кожна підпрограма повинна вирішувати своє власне завдання. Коли відбувається написання програми здається що вирішення рішення проблеми безпосередньо у місці де вона виникла є вірним. Це призводить до того що у одній програмі багато ділянок коду

виконують різні задачі. Гарною альтернативою виступає написання підпрограм під різні задачі наприклад такі як введення даних, наступна ж служить для керування, а третя підпрограма становить загальний метод підпрограми. У той же час підпрограмі введення інформації. Але допускати ситуації, коли підпрограма введення інформації намагається ще і обробити данні, і навіть коли вона намагається керувати проистроєм не можно.

Вони всі включені в нескінченний цикл для роботи всіх написаних підпрограм. Це теж саме що і, періодичний запуск апаратних блоків. Взаємодія частин програм стає з’єднанням, за допомогою глобальних змінних.

Інформувати оператора о “невизначених” ситуаціях буде цикл який описан вище в котрім передбачено блок обробки помилок. В ньому наприклад показуватимуть невірну інформацію отриману з пристрою або клавіатури котрі підключені до мікроконтроллера.

Програмна реалізація планувальника виглядає наступним чином:

```
{
initTaskList();
runTasks();
while (1) {
clearWatchDogTimer();
}
}
```

Спочатку потрібно провести аналіз всіх пристроїв котрі в потрібні в цьому проекті, наступним шагом створити чергу задач, покликати головний цикл з задачами, реалізувати нескінченний цикл. За для уникнення помилок та захистити систему від зависання потрібно застосувати WatchDogTimer.

1.5 Взаємодія задач та розподіл ресурсів

То що було розглянуте вище дає поняття що, на вхід системи можуть надходити задачі абсолютно незалежно. З цього надходить необхідність розподілення ресурсів системи. Реалізувати це можливо двома рішеннями. По перше спосіб FIFO «First In First Out» для його виконання потрібне: у систему на вхід надходить задача, до виконання вона буде отримувати всі ресурси, потім запускається наступна задача по черзі.

По друге існує спосіб карусельної багатозадачності «round robin» при його виконанні виконується поділення часу в системі на кванти «time slice», з наступним розподіленням за алгоритмом між задачами у системі.

Існує декілько моделії диспетчеризації – кооперативна, витісняюча та інші.

Кооперативна багатоздачність – це коли ОС завантажує одночасно у пам’ять два чи більше додатків, з надонням процесорного часу онсновним додатком лише якщо є дозвіл головного потоку.

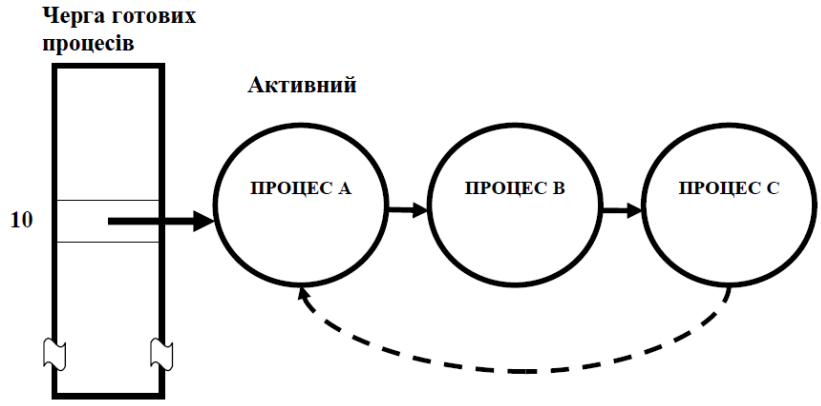

Рисунок 1.5. Кооперативна багатозадачність

Кооперативна багатозадачність це як багатозадачність другого рівня, за рахунок більш досконалих технологій ніж перемикання задач, як у більшості програм. Коли відбувається перемикання задача отримує процесорний час, а задачі які находяться на фоні заморажуються.

Якщо виконання основної задачі відбувається довго, виконання поділяється на невеликі розділи, і як тільки такий розділ буде завершено, процес добровільно поступає чергу, і задача переходить у очікування.

Коли відбувається написання планувальника кооперативного типу необхідно продумати реалізацію кожного паралельного процесу. Інакше є ймовірність того що один процес повність получить ресурси процесора та відбудеться порушення роботи других процесів. Через некоректний розподіл часових ресурсів взагалі деякі фонові задачі не працюють.

Кооперативна багатозадачність має такі перваги: проста реалізація, не великі вимоги до пам’яті, дуже легка синхронізація процесів що взаімодіять між собою

При реалізації витисняючії багатозадачності процесорний час починає ділитися на кванти часу, а вони у цей час виділяються процесором. Свій власний квант мають право використовувати кожний процес, наступним кроком планувальник завершає роботу процеса та передає по черзі квант наступному процесу. Якщо процесу не вдається повністю використати свій квант як приклад робить запит на введення-виводу, він не може бути звершиним негайно, процес будеть негайно заблокованим до завершенням операції, і квант переходить до наступного процесу.

Гарантовантий разподіл часу процесору це головна перевага витісняючої багатозадачності. Гарантоване також що коли один з процесів може зациклитися та перестане реагувати на події котрі адресовані йому, це не презведе до повного збою системи та інші процеси продовжуть отримувати власні кванти часу процесора та використовувати їх.

Але в неї є і недоліки. По перше, це зациклення процесу. Якась задача високого пріоритету зможе захопити любий ресурс котрий є у загальному доступі на великий часовий проміжок, а інши будуть стояти у черзі поки ресури не будть звільнені. Це призводить до порушення роботи програмного забезпечення для керування пристроєм.

По друге, два процеси можуть поділити між собою кванти, що презведе до того що інші задачі не зможуть виконати свій завдання. При цьому самостійно вийти з цього кола вони не зможуть. Тому під час разробці операційної системи необхідно вживати спеціальних заходів для синхронізації взаємодіючих потоків

На рисунку 1.6 зображено роботу витісняючої багатозадачності для задач з різним пріоритетом, де перше це переключення контексту, та запуск задачі, друге код задачі, трете переключення контексту та запуск задачі, четверте код задачі, п'яте переключення контексту та виконання задачі, шосте переключення контексту та продовження виконання фонової задачі.

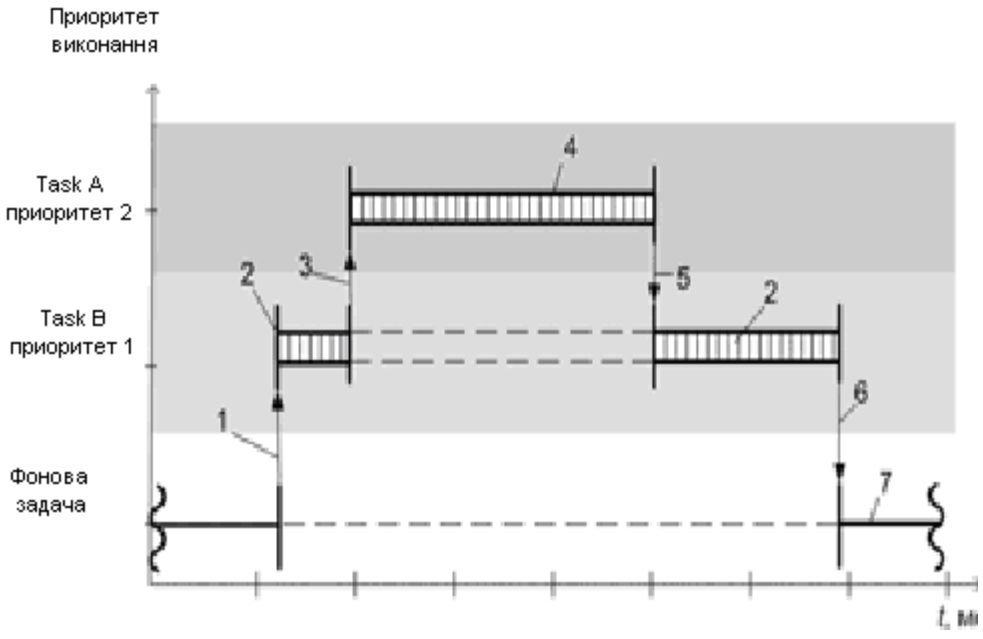

Рисунок 1.6. Витісняюча багатозадачність з різними пріоритетами

У рисинку 1.6 наведена робота витісняючої багатозадачності в котрій є задачі у котрих різні пріорітети. Де перше це переключення контексту . Друге код задачі. Третє це переключення контексту та запуску задачі. Четверте знову код задачі. П'яте переключення контексту і продовження виконання задачі. Та шосте з переключенням контексту продовження виконання фонової задачі.

Рисунок 1.7 наводить схему роботи витісняючої багатозадачності для задачі з однаковим пріоритетом. Де перше інтервал часу котрий закінчено. Друге код різноманітних задач у котрих однаковий пріорітет. Третє запуск наступної задачі з перемиканням контексту. Четверте інтервал часу. Як ми бачемо механізм цієї багатозадачності дуже складний та ресурсоємкий. Мікроконтролери з малим обсягом пам’яті стає складною за рахунок того що необхідно зберігати потоків даних. Як приклад регістер процесу, тоді коли виділення пам’яті під контекстом всіх задач що презведе до нестачі пам’яті таблиці планувальника.

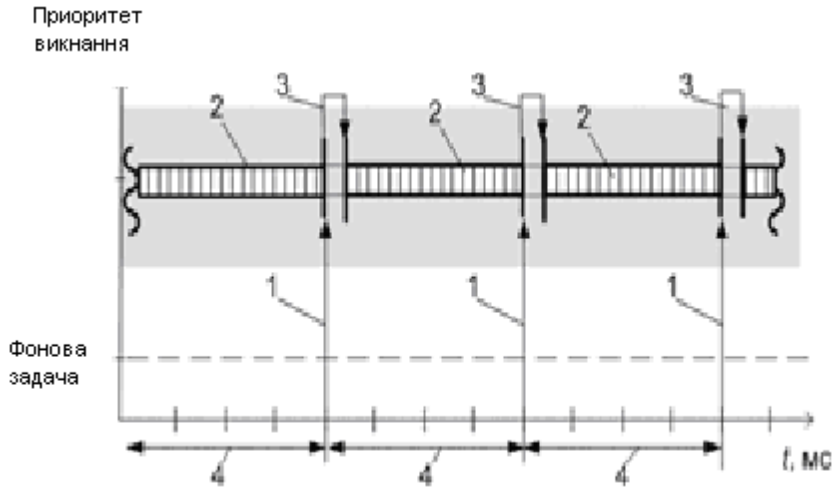

Рисунок 1.7 – Витісняюча багатозадачність з однаковими пріоритетами

1.6. Семафори та м’ютекси

Семафори та м’ютекси використовуються для узгодженого доступу до спільних обсягів ресурсів.

Семафори застосовуються для синхронізації спільних ресурсів. Вони бувають бінарними або рухунковими . Якщо семафор бінарний, то він байтова змінна, це змінна, яка може приймати значення «нуль» чи «одиницю». Коли черга підходить до задачі вона виконується, та семафор захоплює інші задачі, котрім потрібно використовувати спільні ресурси, чекають звільнення. Лічильником є семафор.

При зверненні до спільних ресурсів можуть бути критичні ділянки коду (глобальні зміни, читання даних тощо). Для вирішення подібних ситуацій потрібно використовують механізм взаємовиключення доступу - м'ютексів. М'ютекс - це варіант семафору, який сигналізує про інші дії, пов'язані з ресурсами. Нижче наведено приклад реалізації програми Mutex. 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);
  }
```

Коли м'ютекс захоплен, усі задачі з глобальною змінною, як в приклоді вишче, то вона блокується. М'ютекс після цього звільняється, та інші задачі здобувають доступ до ресурсів.

1.7 Обробники переривань

Крім чіткої послідовності виконання задач за правилами, розглянутими вище, у системі реального часу можуть використовуватися зовнішні преривання різноманітних пристроїв. Однак слід враховувати наступне.

Перш за все, іноді необхідно припинити обробку будь-яких преривань, наприклад, якщо втручання не дозволено (перетворення АЦП). У такому випадку переривання з початку функції вимикаються, зі зніманням заборони, інакше це викликає помилку.Інак ше це може викликати помилку. Це пов’язано з тим, що в цьому коді є критичні єлементи, які краще захищені.

Такі частини коду мають бути якомога коротшими. Їх періодичність також слід максимально зменшити. З метою скорочення часу, що витрачається на обробку преривання операцій, допускаються лише виконувати первинні завдання, наприклад, зчитування первинних даних , а подальші обчислення виконуються в інших, не критичних до часу елементах коду. У цьому випадку самі необхідні дії виконує обробник преривань, у той же час обробка відкладується до того часу поки не будє виконена задача-обробником преривань. Ця обробка називається відкладеною обробкою.

Після завершення обробки переривання процесор повертає керування до планувальника.

Після завершення обробки переривань, починаються виконання перемикання задач. Це робіть системна функція повернення з переривання, за умови встановленого прапора.

1.8 Огляд існуючих ОСРЧ

На цей час існує велика кількість ОСРЧ, для 8,16,32,62 розрядних мікропроцесорів. Частина з них, закінченні ОС та включають у весь перелік функцій.

Таблиця 1.1.ОСРЧ та мікроконтролери, що підтримуються ними

ChibiOS/RT	**Embox**	**FreeRTOS**
PIC32[	ARM	ARM (ARM7, ARM9, Cortex-M0, Cortex-M3, Cortex-M4, Cortex-A)
ARM7	RISC-V	Atmel AVR
Cortex-M0, -M0+, -M3, -M4, -M7	LEON	AVR32
PPC e200zX	MicroBlaze	HCS12
STM8	MicroBlaze	MicroBlaze
MSP430	MIPS	Cortus (APS1, APS3, APS3R, APS5, FPF3, FPS6, FPS8)
AVR	E2K	MSP430

ChibiOS/RT - компактна ОСРЧ для вбудованих систем. Має високу мобілність, маленький об'єм.

Embox – свободна ОСРЧ для вбудованих систем. Оснвна особливість максимальна структированість з можливістью задання параметрів.

FreeRTOS - багатозадачна ОСРЧ, має високу партированість на мікропроцесорні архітектури. Основна задача це робота на масових мікроконтролерах котрі мають малий об’єм оперативної памяті та бистродії.

1.9. Особливості побудови системи в реальному часі

Після проведеного аналізу сформулюємо вимоги які пред'являються до системі реального часу.

1. Вони мають бути вбудовані у великі програмно-апаратні комплекси. При цьому взаємозв'язок з іншими частинами такої системи повинен бути стандартизований і не повинен вносити істотних змін у розробку програмного забезпечення та обладнання при підключенні системи, що розробляється. Для цього в систему, що розробляється, повинні бути включені різні комунікаційні програми та драйвери.

2. Вони повинні вибирати різні датчики для взаємодії із зовнішнім середовищем та вживати заходів контролю, використовуючи різні механізми або сигнали повідомлень.

3. ОСРЧ має будуватися за принципом багатозадачності і водночас допускати витиснення.

4. Обробка всіх вхідних впливів і формування керуючих сигналів повинні здійснюватися в встановлені часові терміни, оскільки допустимі тимчасові відхилення іноді можуть призвести до незворотних наслідків. Тобто максимальний час має бути чітко заданим.

5. При збереженні механізмів синхронізації слід виходити з пріоритетів для різних завдач.

6. Має бути механізми для блокування та комунікації задач, які спільно ділять ресурси мікроконтролера.

РОЗДІЛ 2. ОГЛЯД ТА РЕАЛІЗАЦІЯ АПАРАТНОЇ ЧАСТИНИ

2.1. Платформа Arduino

Arduino це готовий набір, що складається з програмного забезпечення та готового електронного блока. Електронний блок насамперед це печатна плата з вбудованим мікроконтролером та елементами, необхідних для роботи.

Електронного блока Arduino являє собою аналог материнської плати. Але має роз'єм зв'язку з комп'ютером, яким і здійснюється програмування мікроконтролера.

Завдяки використанню мікроконтролерів ATmega зникає потреба в спеціальних програматорів. Також дуже важливим плюсом є програмне забезпечення для створення програм управління. Воно поєднало в собі найпростіше середовище розробки та мову програмування, що є варіантом мови C та C++ для мікроконтролерів.

2.2 Технічне завдання для апаратної частини

Задля реалізації системи управління об’єктом на основі ОСРЧ сформуємо технічне завдання з двух паралельно опитуваних датчиків.

Сформоване наступне технічне завдання: об’єкт повинен автономно проходити через ”випадковий” лабіринт з перешкодами. Об’єкт має самостійно виходити з лабіринту з маневруванням від перешкод.

Об’єкт за допомогою заданої логіки та інфрачервоних датчиків повинен вирішувати напрямок руху, аналізуючи місцевість для прокладання шляху, минаючи перешкоди. Перешкоди реалізовані завдяки магнитів. Перешкода виявляється завдяки датчика Холла. При виявленні перешкоди, міняється траікторія руху об’єкта для обїзда її.

2.3 Модель Об’єкта

За для реалізації об’єкта, котрий буде виконувати рух за технічним завданням, потребно 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 перешкод.

2.4 Вибір апаратної платформи і сенсорів

Після огляду сімейства Arduino, було прийняте рішення використовувати її як платформу модуля управління.

Платформи сімейства Arduino включає в себе велику їх різноманітність, включно з їх різними мікроконтролерами. Більшість з них знаходяться у системі AVR, але тако ж є і на ARM.

За рахунок того що у плтах Arduino не має вбудованих моніторів, датчиків чи кнопок, за винятком кнопки для перезавантаження. Для компенсації цього плати мають богато зовнішніх датчиків, плат розширення та інших які можливо підключити до плати.

Після проведеного аналізу було прийнято рішення використовувати як модуль управління платформу Arduino Uno як пристрій керування двигунами та модулем обробки даних від датчиків.

Arduino Uno це базова плата сімейства Arduino. Вхідна напруга -5В та діапазон напруги живлення від 7В до 12В. Кількість аналогових виводів сягає 6 та 14 цифрових, з можливістю підключення плати розширення. Завдяки відносно маленьким розмірам з кількостью виводів цє дуже зручно для роботи, коли потрібна оптимально велика кількість єлементів підключення та при цьому мати невеликий розмір.

Для реалізації об’єкта обрано такі датчики: інфрачервоний датчик, датчик Холла.

Інфрачервоний датчик працює так: зі світлодіода випускається інфрочервоний сигнал що відбивається від перешкод та ловится фоторезистором. Відстань виявлення перешкод від декількох сантиметрів до десятків сантиметрів. Фотографія датчика.

Рис 2.1. Інрфачервоний датчик перешкод

Дальність ефективного виявлення перешкод ІЧ датчика до 30 см яка залежить від декількох факторів: точна генерація випромінювання сигналу, потужність випромінювання світлодіодів, особливість поверхні виявлення перешкод.

За для фіксації перешокод у лабіринті був обран датчик Холла. Він працює за приципом зміни вихідної напруги в залежності, присутнє чи неприсутнє мгнітне поле. Прешкодами будуть мгніти, що разтощовані у лабіринті в різних містах. Коли датчик знаходить магнітне поле перешкоди то міняє напругу струму сигналу входу.

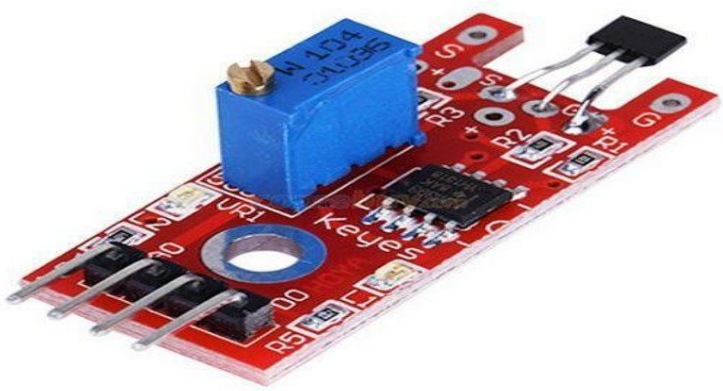

Рисунок 2.2. Датчик Холла

Рух об’єкту повинен здійснюватися за рахунок чотирьох двигунів.Плати Arduino, крім спеціалізованих, не підтримають керування двигунами постійного струму, тому було прийняте рішення застосувати схемну реалізацію драйвера двигунів на мікросхемі L298N. Її запас максимального струму є первагою. Далі на рисунку 2.3. буде наведено схему підключення драйвера двигунів до платформи Arduino Uno.

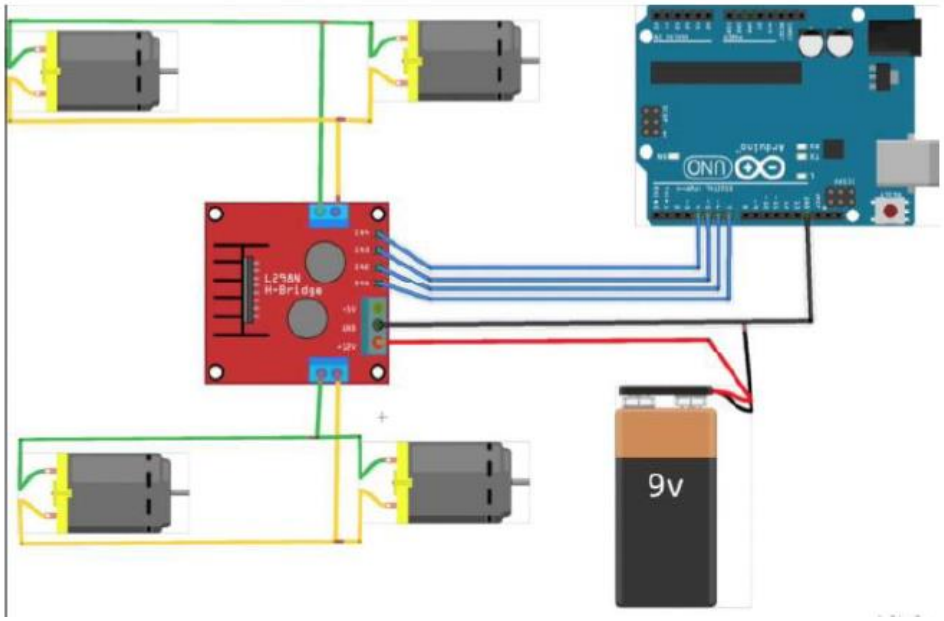

Рисунок 2.3. Схема підключення двигунів до драйверу двигунів та до платформи Arduino Uno

2.5 Алгоритм руху об'єкта

Три ІЧ-датчики розташовані на передній частині об’єкта, для виявлення ммеж лабіринту. За інформацією котру мікроконтролер получає від ІЧ-датчиків, він задає напрямок руху та формує сигнали на управління двигунами. Також у корпусі є датчик Холла, за для виявлення перешкод. Об’єкт починає рухатися лабіринтом, одночасно аналізуючи інформацію, отриману з датчиків. Коли об’єкт під'їжджає до стіни, відповідний датчик виявляє бар’єр і вирішує повернутися в протилежному напрямку, за для уникнення зіткнення. Керування рухом включає два компоненти логічна “нуль” для реверсу руху, та “одиниця” для руху вперед. За допомогою ШИМ - широтно імпульсної модуляцої реалізоване управління швідкістю.

Коли датчик Холла виявляє перешкоду, об'єкт дає реверс ходу та повертає вправо, та продовжує свій шлях.

РОЗДІЛ 3. ОГЛЯД ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

3.1 Arduino IDE

Среда разробки Arduino складєтся з вбудованого редактора програмного кода, області повідомлень, консолі виведення тексту, інструментальна панель в котрій є кнопки для бистрого доступу команд які часто використовуються.

Програма пишеться в текстовом редакторі у якого є інструменти вставки/вирізки, пошука та заміни текста. Коли відбувається збереження та експорт проекту то в області повідомлень з’являються пояснення чи помилки.

Консоль показує повідомлення безпосередньо від Arduino, в які входить повний звіт о помилках та інша інформація. Усі кнопки яки разташовані на паналі дозволяють швидко виконувати усі необхідні дії.

Verify/Compile –Код перевіряється на ошибки та компілюється.

Stop – Припинення моніторінгу шини

New – Створити новий скетч.

Open – Відкриває меню для доступу ко всім скетчам.

Save – Збереження скетчу.

Upload to I/O Board = Компеляция и завантаження коду до Ардуіно

Serial Monitor– Відкриття послідовності шини.

Допоміжни команди зроблені у пять меню: File, Edit, Sketch, Tools, Help.

Edit:

- Copy for Discourse

Копіює до буфера обміну відповідний для розміщення на форумі код скетчу з виділенням синтаксису.

- Copy as HTML

Копіює код скетчу в буфер обміну як HTML код для розміщення на веб-сторінках.

Sketch:

- Verify/Compile

Перевірка скетчу на помилки.

- Import Library

Додає бібліотеку до поточного скетчу, вставляючи директиву #include у код скетчу. Детальна інформація в описі бібліотек нижче (Libraries).

- Show Sketch Folder

Відкриває папку, що містить файл скетчу, на робочому столі.

- Add File...

Додає файл до скетчу (файл буде скопійовано з поточного розташування). Новий файл з'являється у новій закладці у вікні скетчу. Файл можна видалити зі скетчу за допомогою меню закладок.

Tools:

- Auto Format

Ця опція оптимізує код, наприклад, вибудовує в одну лінію по вертикалі, що відкриває і закриває дужки і поміщає між ними затвердження.

- Board

Вибір платформи. Список із описом платформ наводиться нижче. Serial Port Меню містить список послідовних пристроїв передачі даних (реальних та віртуальних) на комп'ютері. Список автоматично оновлюється автоматично при відкритті меню Tools.

- Burn Bootloader

Це меню записує завантажувач Bootloader на мікроконтролери. Це може не знадобитися, але необхідне якщо зявится новий ATmega з відсутнім завантажувачем. Першою и головною рекомендацією є вірний вибір платформи у меню. Якщо використовується AVR ISP то необхідно вибрати необхідний програматор порта.

Однією з самих головних меню є Бібліотека бо завдяки їй можливе додавання додаткової функціональності скетчам, на приклад, коли відбувається робота з оборобкою даних чи апаратною частиною. Декілько деректив разміщуються на початку скетчу з подальшою компіляцією скетча з бібілотеками. За для застосування бібліотек необхідно додаткова пам’ять. Певна частина бібліотек знаходится у середовищі разробки, інші завантажуються з сторинніх ресурсів.

Також є моніторинг послідовності шини який відображає дані які посилаються в платформу Arduino.

3.2 Мова програмування

Arduino C являє собою мову C++ з фреймворком Wiring. Має деякі відмінності щодо написання коду, компілювання та збирання коду відбувається за допомогою avr-gcc, з деякими особистостями, що полегшують написання програм.

3.3 Технічне завдання для програмної частини

Для реалізації багатозадачності рухомого об’єкту функціональна схема ОСРЧ побудована трьохрівневою:

1-й - рівень Додатків з наступними функціями:

- управління двигунами;
- об’їзд перешкоди по сигналу датчика Холла;
- розрахунок логіки руху;
- включення режиму економії енергії

2-й – рівень ОСРВ (рівень прикладних програм):

- модуль Планувальника:
    - задача 1;
    - задача 2;
    - задача 3;
    - задача 4;
- модуль виконання задач за Перереванням;

3-й – рівень Апаратний:

- двигуни;
- АЦП (аналогово-цифровий перетворювач) обробки сигналів датчиків;
- датчик Холла;

- ІЧ датчик.

Виконання зазначених 4-х задач, тобто реалізація багатозадачності, відбувається шляхом розділення коду. Код розбивається на частини, які виконуються послідовно та безперервно. Кожна з задач системи виконується відповідно стану системи, саме як кінцевий автомат.

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

– управління двигунами;

– об'їзд перешкоди;

– розрахунок оптимального руху;

- визначення моментів часу включення режиму економії електроенергії та переходу мікроконтролера в стан сну.

На рівні ОСРЧ було запропоновано два модуля - це програма Планувальника і програму обробки Переривань. Програма Планувальник має забезпечувати послідовність виконання задач, тобто формувати чергу. А переключення по контексту програмного коду відповідної задачі з черги забезпечується диспетчером. Таким чином відбувається послідовне виконання черги задач за реальним плином часу.

Модуль виконання задач за Перериванням забезпечує видачу на чергу задач відповідного параметра 0 або 1, що дозволяє виконувати задачу або не виконувати, тобто працює як бінарний світлофор.

Апаратний рівень функціональної схеми забезпечується опитуванням датчиків Холла та ІЧ, перетворенням цих сигналів в цифровий вигляд для опрацюванням і формування управляючих команд на двигуни.

3.4 Конкретизація задач ОСРЧ

Враховуючи запропоновану функціональну схему ОСРЧ конкретизуєм задачі управління рухомим об’єктом:

- отримання від датчика Холла аналогового сигнала;

- отримання від ІЧ датчика аналогового сигнала;

- перетворення отриманних аналогових сигналів в цифровий вигляд;

- визначення оптимального руху на основі аналізу перетворених сигналів датчиків.

- управління швидкістю обертання двигунів та реалізація руху (поворот вправо, вліво, вперед, назад) і розрахунок для ШІМ.

Спробуэмо визначити послідовність та першочерговість виконання задач. Переміщення в лабіринті допускає практично одночасне отримання об’єктом сигналів від датчика Холла та від ІЧ датчиків. Так як процесор тільки один, то одночасно обробити ці сигнали ми не можемо. В той же час, обидва сигнала край важливі для об’єкту і ігнорувати кожний з них ми не можемо. Тому для однозначності дій системі управління були запропоновані наступні приорітети обробки сигналів або вирішення задач:

1 – обробка сигналів від датчика Холла (тобто вчасне виявлення перешкоди);

2 – забезпечення руху назад при виявленні перешкоди;

3 – обробка сигналів ІЧ датчиків та АЦП-перетворення;

4– визначення та забезпечення оптимального руху.

3.5 Програма управління двигуном

Програма управління двигуном написано шляхом введення 6 цілочисельних глобальних змінних. Ці змінні забезпечують включення та виключення моторів і швидкість їх обертання.

Лістинг 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
```

Для того, щоб об’єкт рухався у просторі лабіринту у різних напрямах відповідно сигналів датчиків були розроблені наступні, так звані, базові функції:

Setup_motor_system – функція назначає глобальним змінним номера портів Arduino і перехід відповідного порту в режим виводу даних;

speedPinR – функція забезпечує зміну швидкості руху об’єкту і потужність на основі ШІМ.

Управління роботою двигунів забезпечено розробкою таких функцій: рух вперед, рух вправо, рух вліво, рух назад. Вони наведені у листингу 3.2.

Лістинг 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 зміни швидкості обертання двигунів використана функція set_Motorspeed, аргументи якої задають параметри швидкості.

Лістинг 3.3

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

```
analogWrite(speedPinR, speed_R);
  }
```

3.6 Програма моніторингу сигналів датчиків

Зчитування сигналів з датчиків реалізовано розробленими функціями. Лістинг 3.4 наведені функція задіяння датчика Холла та функція, яка опитує сенсор. Функція реалізує задачу об'їзду об’єктом перешкоди.

Лістинг 3.4

```
#define MFsensor D2
void task1_read_MFsersor()
{
go_Back(); // Обережно перешкола! Їдемо назад та повертаємо направо.
delay(10);
go_Right();
}
```

Функція моніторингу чи опитування датчика Холла запускається за станом переривання. Функція опрацювання переривання наведена нижче:

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

Рух по лабіринту відбувається за допомогою функції, яка ініціалізує та моніторить ІЧ сенсори. Ця функція наведена в листінгу 3.5 та вирішує задачу опитування ІЧ сенсорів та ще виконує аналогово-цифрове перетворення отриманих від сенсорів сигналів.

Але специфіка виконання аналогово-цифрового перетворення сигналів ІЧ сенсорів що воно не може буте відкладено чи припинено за будь яким переріванням. Для коректного виконання АЦП на початку функції з допомогою функції mutexLock(); вводится м’ютекс. З допоиогою данної функції блокується переривання потоку, також заблоковується виконная переривання. Після завершення АЦП сигналів ІЧ мьютекс вже не потрібен. Переривання можут виконуватись і тому використовуєм функцію розблокування м'ютекса 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();
  }
  }
```

3.7 Програма реалізації керування рухом

Технічне завдання на апаратну частину є основою для розробки модуля управління рухом об'єкту. Напрямок руху змінюється на підставі аналізу даних ІЧ датчиків. Граф-схема алгоритму керуванням рухом об'єкта наведено на рисунку 3.2.

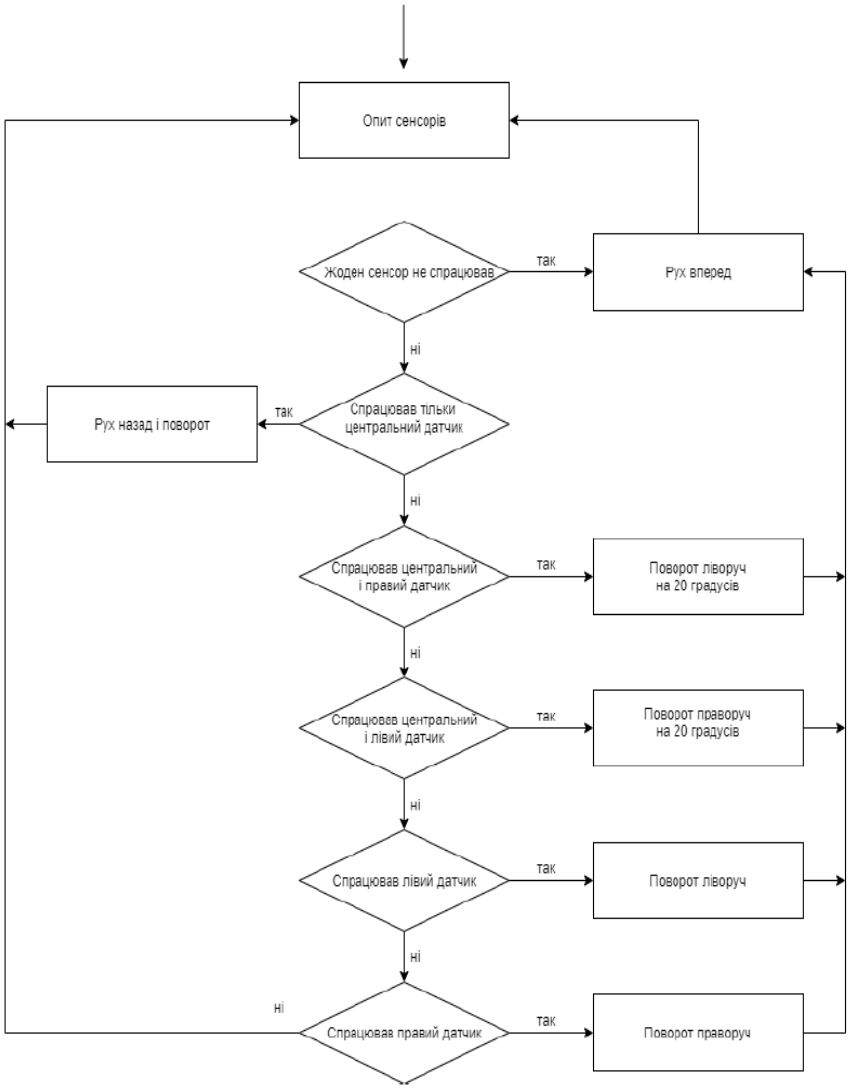

Рисунок 3.2. Граф-схема алгоритму керування рухом

Граф-схема алгоритму керування рухом дає можливість побудувати модель керуючого автомату. На рисунку 3.3. наведений Граф можливих переходів автомату. Вхідні сигнали позначені ребрами графу Хі, а вихідні сигнали записані ребрами графа Уі. Ці сигнали виконуються при переході від одного стану до іншого.

Автомат управляє рухом об’єкта в автономному режимі. На старті приймається стан руху «вперед» і він діє до першого спрацювання одного з датчиків. При виявлені будь якої перешкоди об’єкт виконує поворот у протилежний бік.

Опис шести станів руху автомату:

С0 – вперед;

С1 – наліво на кут 20º;

С2 – наліво на кут 90º;

С3 – назад;

С4 – направо на кут 20º;

С5 – направо на кут 90º.

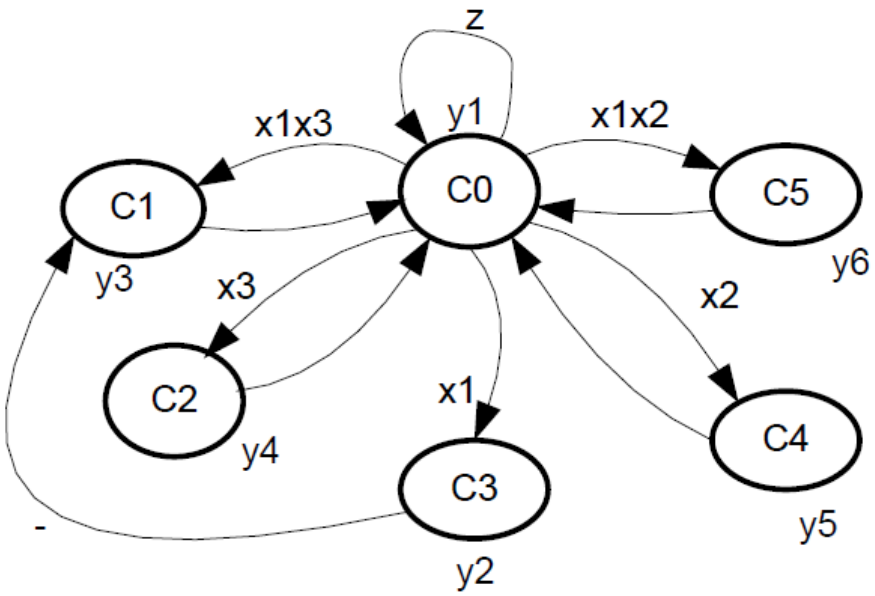

Рисунок 3.3. Граф можливих переходів автомата

Опис вхідних сигналів автомата:

х1 – відпрацював центральний датчик;

x2 – відпрацював лівий датчик;

x3 – відпрацював правій датчик;

z – сформований сигнал опитування датчиків.

Опис вихідних впливів автомата:

y1 – видача команди «вперед»;

y2 – видача команди «назад»;

y3 – видача команди «наліво на кут 20º»;

y4 – видача команди «наліво на кут 90º»;

y5 – видача команди «направо на 20º»;

y6 – видача команди «направо на 90º».

Програмний код руху згідно графу переходів автомату наведена у листингу 3.6

Лістинг3.6

```
void task3_auto_tarcking()
{ 44
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);
}
}
}
}
```

3.8 Програмний модуль Планувальника

Програмний модуль Планувальника виконаний у вигляді системи з циклічним опитуванням. Здійснюючи послідовне опитування кожної задачі Планувальник визначає необхідність виділення для неї процесорного часу. Якщо це необхідно, то процесорний час надається. По закінченню задачі операційна система надає доступ до наступної задачі, яка вимагає процесорного часу.

Програмний модуль Планувальника наведений в листингу 3.7.

Лістинг 3.7

```
taskSwitchinterrupt(uint32_t taskIndex) {
```

Якщо стоїть мьютекс то продовжуємо виконання задачі

```
if (getMutexStatus() == MUTEX_LOCK)
return;
pushCurrentTask(taskIndex);
if (getTaskCnt() < taskIndex) {
popNextTask(taskIndex + 1);
} else {
popNextTask(0);
}
}
```

У лістингу 3.8 наведена програма основної функції. Перш за все формується список задач (покажчики на функції). Потім вмикається таймер для відліку квантів часу. Таким чином створюється черга задач та відбувається виклик основного циклу з задачами.

Лістинг 3.8

```
void loop() {
// список задач (показчики 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 с задачами

while (1) {

wait(100);

clearWatchDogTimer();

}

}
```

РОЗДІЛ 4. ТЕСТУВАНЯ ПЗ

Тестування системи керування рухом об’єктом має метою визначення якісних та кількісних оцінок таких показників:

– регульованість швидкістю руху;

– задовільна точність руху;

– здатність об’їзду перешкод;

– повторюваність проходження лабіринту;

– відсутність відказів системи

– надійність керування.

За результатами тестування. Об'єкт проходить лабіринт точно. Рух об'єкту проходив без зупинок, з маневруванням від перешкод. Прі багатократних випробуваннях з різними начальними умовами об'єкт впевнено і точно проходив лабіринт, показуючи лише різний проміжок часу на проходження лабіринту.

Тестування виконувалось у Windows SandBox для безпечного тестування програмного забезпечення та самого мікроконтроллера.

Також були підключені:

- бібліотеки генератора випадкових чисел;
- емулятор зовнішніх подій;
- віртуальний системний таймер.

А замість управління реальними виконавчими пристроями було здійснено текстовий вивод керуючих сигналів на консоль.

На Рисунку 4.3 зображені результати моделювання, а саме, наведені перемикання між задачами: опитування датчиків, формування керуючих сигналів для двигунів.

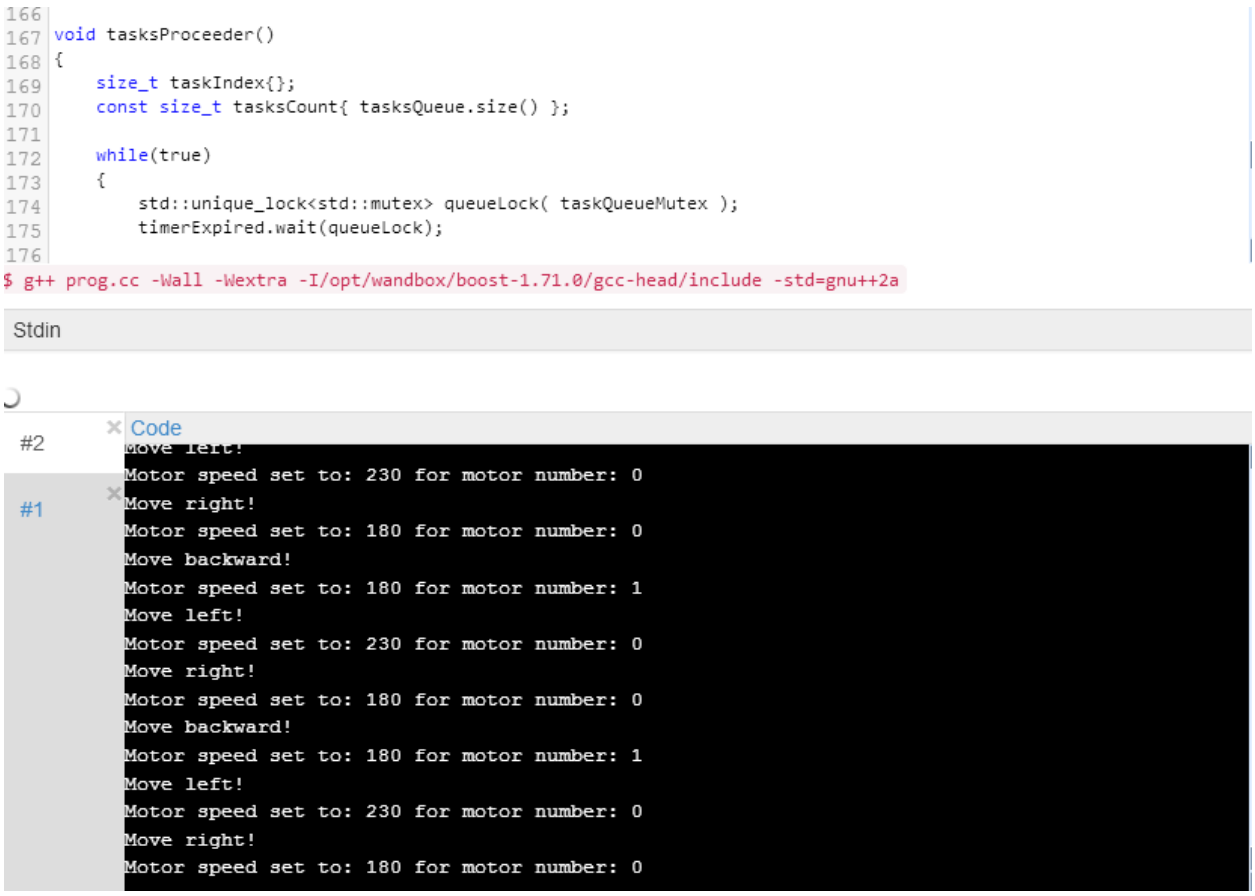

Рисунок 4.3 – Результати моделювання в середовищі Windows Sandbox

ВИСНОВКИ

Розроблено - програмні модулі управління рухом, опитування датчиків і програмний модуль планувальника. Усі ці програми були реалізовані за допомогою планувальника ОСРЧ FreeRTOS.

1. Сформульовані вимоги для реалізації багатозадачності з використанням FreeRTOS.
2. Сформовано технічне завдання і на його підставі були розроблені апаратна і програмна моделі.
3. Розроблена модель об’єкта, здійснено вибір апаратної платформи і необхідних сенсорів.
4. Визначено як операційна система реального часу дозволяє програмісту зосередити свої зусилля на вирішенні конкретних завдань (алгоритмічних, математичних тощо), не відволікаючись на другорядні задачі та полегшує написання програмного коду.
5. Реалізовано програмне забезпечення в наступні етапи:
    1. виявлення перешкод, тобто обробка події від датчика Холла;
    2. рух назад при виявленні перешкоди;
    3. читання ІК датчиків та АЦП-перетворення;
    4. розрахунок оптимального руху.

Визначено - операційна система реального часу дозволяє програмісту зосередити свої зусилля на вирішенні конкретних завдань (алгоритмічних, математичних і т.п.), не відволікаючись на другорядні задачі та облегшити написання програмного коду в декілько разів. Також було визначено що написане програмне забепечення показало єфективність та надійність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. National Instrument What is a Real-Time Operating System (RTOS)? / Дата обращения: 24 ноября 2019. - http://www.ni.com/white-paper/3938/en/

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

3. Рыжов. Д: Разработка систем реального времени с использованием UML и каркасов приложений: http://www.myshared.ru/slide/111925

4. Микушин, А.В. Цифровые устройства и микроконтроллеры: учебное пособие/ А.В. Микушин, А.М. Сажнев, В.И. Сединин. – СПб.: БВХПетербург, 2010 – 832.

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

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

7. Dedicated Systems Experts, QNX® RTOS 6.1// Dedicated Systems– 2002. – 97 p.

8. Блискавицкий, А. [Электронный ресурс]: Операционные системы реального времени/ А. А. Блискавицкий, С. В. Кабаев. – Средства и системы компьютерной автоматизации. Режим доступа.

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

10. Ярнольд, Стюарт. Arduino для начинающих : самый простой пошаговый самоучитель / Стюарт Ярнольд ; [пер. с англ. М. Райтман]. — Москва : Эксмо, 2017. — 256 с. — (Электроника для начинающих).

11. Андрей Курниц FreeRTOS — операционная система для микроконтроллеров-2011.

12. Dedicated Systems Experts, QNX® RTOS 6.1// Dedicated Systems . – 2002. – 97 p

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

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

15. Блискавицкий, А: Операционные системы реального времени/ А. А. Блискавицкий, С. В. Кабаев. – Средства и системы компьютерной автоматизации. Режим доступа: http://www.asutp.ru.

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Багатозадачність на Ардуїно на основі операційної системи реального часу FreeRTOS

Виконав студент 4 курсу
Карпенко Богдан Анатолійович

Керівник роботи

Корецька Вікторія Олександрівна,
доцент кафедри інженерії програмного забезпечення

Київ – 2022

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Об’єкт дослідження** – процес побудови складних багатозадачних систем.
- **Предмет дослідження** – дослідження операційних систем реального часу для організації багатозадачності на Arduino.
- **Мета дослідження** - покращення організації багатозадачності на Arduino за допомогою розробленої програми на основі операційної системи реального часу FreeRTOS.

АНАЛОГИ ОСРЧ FREERTOS

Порівняння ОСРЧ	**ChibiOS/RT**	**Embox**
Підтримка платформ:	ARM7,ARM9,Cortex-M0(-M0+, -M3, -M4, -M7 PPC e200zX),STM8,MSP430,AVR,PIC32.	SPARC v8(LEON3),ARM, RISC-V, MicroBlaze, MIPS,PowerPC,E2K.
Типи ядер:	Мікроядро	Екзоядро
Особливості програмного коду:	Компактність та ефективність	Максимальна структурованість

ВИСНОВКИ

1. Сформульовані вимоги для реалізації багатозадачності з використанням FreeRTOS.
2. Сформовано технічне завдання і на його підставі були розроблені апаратна і програмна моделі.
3. Розроблена модель об’єкта, здійснено вибір апаратної платформи і необхідних сенсорів.
4. Визначено як операційна система реального часу дозволяє програмісту зосередити свої зусилля на вирішенні конкретних завдань (алгоритмічних, математичних тощо), не відволікаючись на другорядні задачі та полегшує написання програмного коду.
5. Реалізовано програмне забезпечення в наступні етапи:
   - виявлення перешкод, тобто обробка події від датчика Холла;
   - рух назад при виявленні перешкоди;
   - читання ІК датчиків та АЦП-перетворення;
   - розрахунок оптимального руху.

Дякую за увагу!