Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

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

_____Універсальний цифровий ключ на основі технології RFID_____

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

_____Пашков Д.О._____
(прізвище, ініціали)

Спеціальність *123 Комп'ютерна інженерія*

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

Освітня програма_____

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

Керівник роботи доц. Хаханова Г.В.
(посада, прізвище, ініціали)

Допускається до захисту

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

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва

ЗАТВЕРДЖУЮ:

Зав. кафедри
(підпис)

«____»_____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пашкову Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Універсальний цифровий ключ на основі технології RFID

затверджена наказом по університету від «_04_» 11 2021 р. № _1635 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23.12.2021

3. Вихідні дані до роботи Стандарти та протоколи безконтактної технології RFID
Мікроконтролер ATtiny85
USBASP - програматор
Дісплейний модуль OLED
Мова програмування C

4. Перелік питань, що потрібно опрацювати в роботі
аналіз математичного апарату та специфікацій технолог
проектування та реалізація апаратної частини проекту
проектування та реалізація програмної частини проекту
аналіз отриманих результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _20 слайдів_____
_____
_____
_____
_____

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання 20.10.2021_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	20.10.2021 - 25.10.2021	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	25.10.2021 - 10.11.2021	
3	Розробка моделі пристрою керування	10.11.2021 - 20.11.2021	
4	Розробка електричної схеми пристрою	20.11.2021 - 30.11.2021	
5	Розробка програми для мікроконтролера	30.11.2021 - 10.12.2021	
6	Проведення випробування пристрою	10.12.2021 - 15.12.2021	
7	Оформлення пояснювальної записки	15.12.2021 - 20.12.2021	
8	Перевірка виконаного проекту керівником,	20.12.2021 - 23.12.2021	
9	Захист проекту	23.12.2021 - 28.12.2021	

Студент _____ _____
(підпис)

Керівник роботи (проекту) _____ _____доц. Хаханова Г.В.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 80 сторінок, 16 рисунків, 5 таблиць та 7 джерел за переліком посилань.

МІКРОКОНТРОЛЕР, ДІСПЛЕЙНИЙ МОДУЛЬ, RFID, СКУД, ATTINY85, КОМП’ЮТЕРНА СИСТЕМА

У роботі розглянуті питання створення універсального цифрового ключа. Проаналізовані принципи роботи та різні стандарти технології RFID, досліджено математичний апарат необхідний для реалізації апаратної частини проекту.

Проведено проектування та реалізацію універсальний цифровий ключ на основі компактного 8-бітного AVR мікроконтролера фірми Atmel, сімейству tiny, модель ATtiny85 з використанням технології RFID для систем керування та управління доступом. В рамках даної системи було розроблено програму мовою С та реалізовано алгоритм емуляції RFID-карток в системах контролю та управління доступом.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЛП– арифметико-логічний пристрій;

АЦП – аналого-цифровий перетворювач;

ПЗ – програмне забезпечення

ООП – об'єктно орієнтоване програмування

СКУД – системи контролю та управління доступом

ОЗУ –запам'ятовуючі пристрої ;

RISC – reduced instruction set computer (зменшений набір інструкцій комп'ютера)

CISC – complex instruction set computing (повний набір інструкцій комп'ютера)

ISP – In System Programming (у системному програмуванні)

LED – Light emitting diode (світловипромінюючий діод)

OLED – organic light emitting diode (органічний світлодіод)

RFID – Radio Frequency Identification (радіочастотна ідентифікація)

FSK – Frequency-shift keying (модуляція частотним зсувом)

TWI – Two-Wire Interface (дво-дротовий інтерфейс)

SPI – Serial Peripheral Interface (послідовний периферійний інтерфейс)

BIN – Binary (двійковий)

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

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

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

ЗМІСТ

ВСТУП

Системи контролю і управління доступом, частіше іменовані СКУД, набули широкого поширення на різних об'єктах, де необхідно забезпечити санкціонований вхід / вихід людей і в'їзд / виїзд транспорту в охоронювані зони.

Система контролю і управління доступом являє собою складний комплекс технічних пристроїв, програмного забезпечення і певних заходів, який дозволяє отримувати, обробляти і зберігати інформацію про кожному проходженні точки доступу і забезпечити строго санкціонований доступ в приміщення / на територію відповідно до рівня доступу кожного співробітника. Крім того, установка СКУД дозволяє контролювати приміщення кожного працівника по території об'єкта, вести облік робочого часу. За допомогою системи контролю доступу також досягається:

- ідентифікація осіб, що мають право доступу;
- розмежування доступу до різних приміщень;
- керування автоматичними режимами;
- реєстрація часу перебування особи на об’єкті;
- обробка інформації та ведення статистики.

Впровадження СКУД дозволяє організувати безпеку та контроль об’єктів без залучення великої кількості працівників охорони та стабільну роботу автоматизованих систем у режимі цілодобово (наприклад, банкоматів, які встановлено в окремих приміщеннях відділень).

Незалежно від технічних особливостей, кожна СКУД має центральний контролер. На основі отриманих даних саме він приймає рішення щодо надання чи заборони доступу. Залежно від типу системи, контролер може працювати автономно чи в об’єднанні з іншими під керуванням головного комп’ютера. Для гарантованої безперервної роботи контролер забезпечено блоком резервного живлення або власним акумулятором. До технічної реалізації контролера нема окремих вимог, та вони варіюються від потреб проекту. Наприклад, якщо в

нашому проекті ми хочемо віддати перевагу компактності обраної системи, а не її продуктивність, можно обрати окремий клас пристроїв під загальною назвою «мікроконтролери».

Мікроконтролер, або однокристальна мікроЕОМ – виконана у вигляді мікросхеми спеціалізована мікропроцесорна система, що включає мікропроцесор, блоки пам'яті для збереження коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники, компаратори, АЦП та інші). Вони широко використовується для керування електронними пристроями. Мікроконтролер можно ємко описати як однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів. Побудова комп’ютерної системи навколо мікроконтролера надає ряд суттєвих переваг наведених нижче:

- значно підвищується гнучкість системи;
- істотно знижується собівартість;
- знижується час розробки та модифікації;
- підвищується надійність системи за рахунок скорочення кількості корпусів і з’єднань;
- знижується енергоспоживання готової системи.

1 ОПИС ОБРАННОЇ СИСТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Мікроконтроллер ATtiny85

Комп’ютерна система у данному курсовому проекті реалізована на основі мікроконтролера сімейства tiny, модель ATtiny85.

ATtiny85 – це 8-бітний AVR мікроконтролер фірми Atmel, представник сімейства tiny. Мікроконтролери даного сімейства, як випливає з його назви, є молодшими в лінійці AVR: у них менше число ліній введення-виведення, менший обсяг пам'яті і обмежений набір периферійних пристроїв у порівнянні з мікроконтролерами mega / XMEGA. Але це окупається їх меншою варті стю і малими розмірами. Крім того, tiny мікроконтролери мають ту ж продуктивність, що і старші мікроконтролери сімейства mega. Нижче, на рисунку 1.1 наведено зображення данного мікроконтролера.

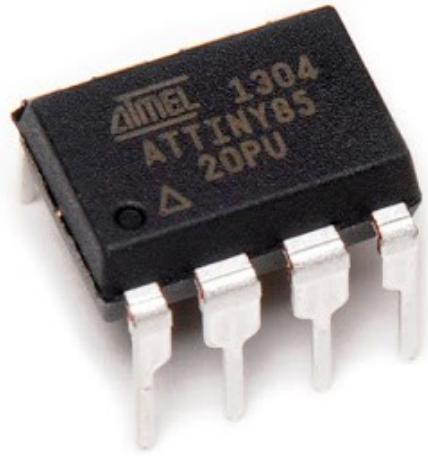

Рисунок 1.1 – Зображення мікроконтролера ATtiny85

Мікроконтролер має наступні характеристики:

– FLASH пам’ять програм – 8Кб;

- ОЗУ – 512 байт;
- незалежна пам'ять EEPROM - 512 байт;
- тактова частота – до 20МГц;
- Universal Serial Interface – універсальний послідовний інтерфейс. Може використовуватися в двохпроводном (I2C / TWI) і трипровідні (SPI) режимі;
- чотирьох канальний десятирозрядний АЦП;
- аналоговий компаратор;
- два восьмибітних таймера-лічильника;
- сторожовий таймер;
- напруга живлення від 2.7В до 5.5В.

Це робить їх відмінним вибором для створення пристроїв, що не вимагають широкого набору периферії, де розмір і ціна мікроконтролера мають значення: tinyAVR знаходять застосування в портативних навігаторах, плеєрах, телефонах, спортивних гаджетах, побутовій техниці, електронних іграшках, пультах дистанційного керування, інтелектуальних датчиках і в багатьох інших пристроях [1, 4, 5].

Також однією за важливих характеристик мікроконтролер є кількість виводів. Мікроконтролер ATtiny85 має вісім виводів, шість з яких можуть використовуватися як лінії введення-виведення [1, 5]. Нижче на рисунку 1.2 наведена схема розташування та використання виводів мікроконтролера.

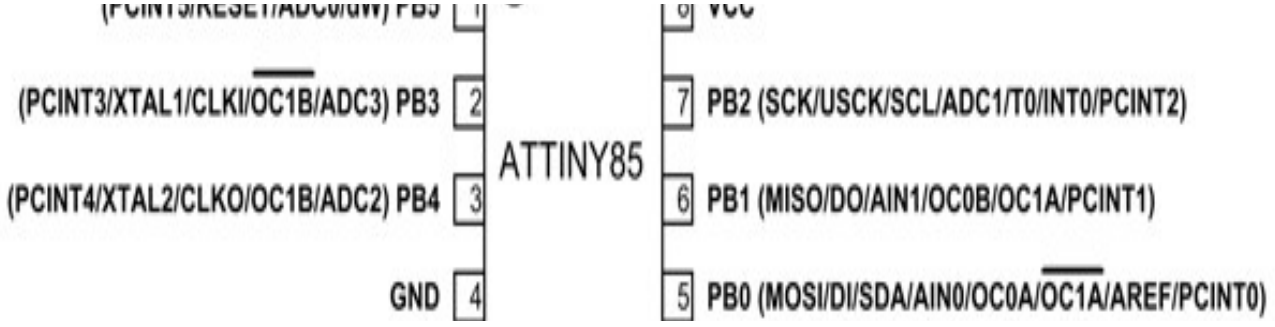

Рисунок 1.2 – Схема розташування виводів мікроконтролера ATtiny85

ATtiny85 – виконано за вдосконаленою AVR RISC-архітектурі (рисунок 1.3). За рахунок виконання більшості інструкцій за один машинний цикл мікроконтролер досягає продуктивності в один мільйон операцій в секунду при тактовій частоті 1МГц, що дозволяє розробнику оптимізувати енергоефективність і швидкодію.

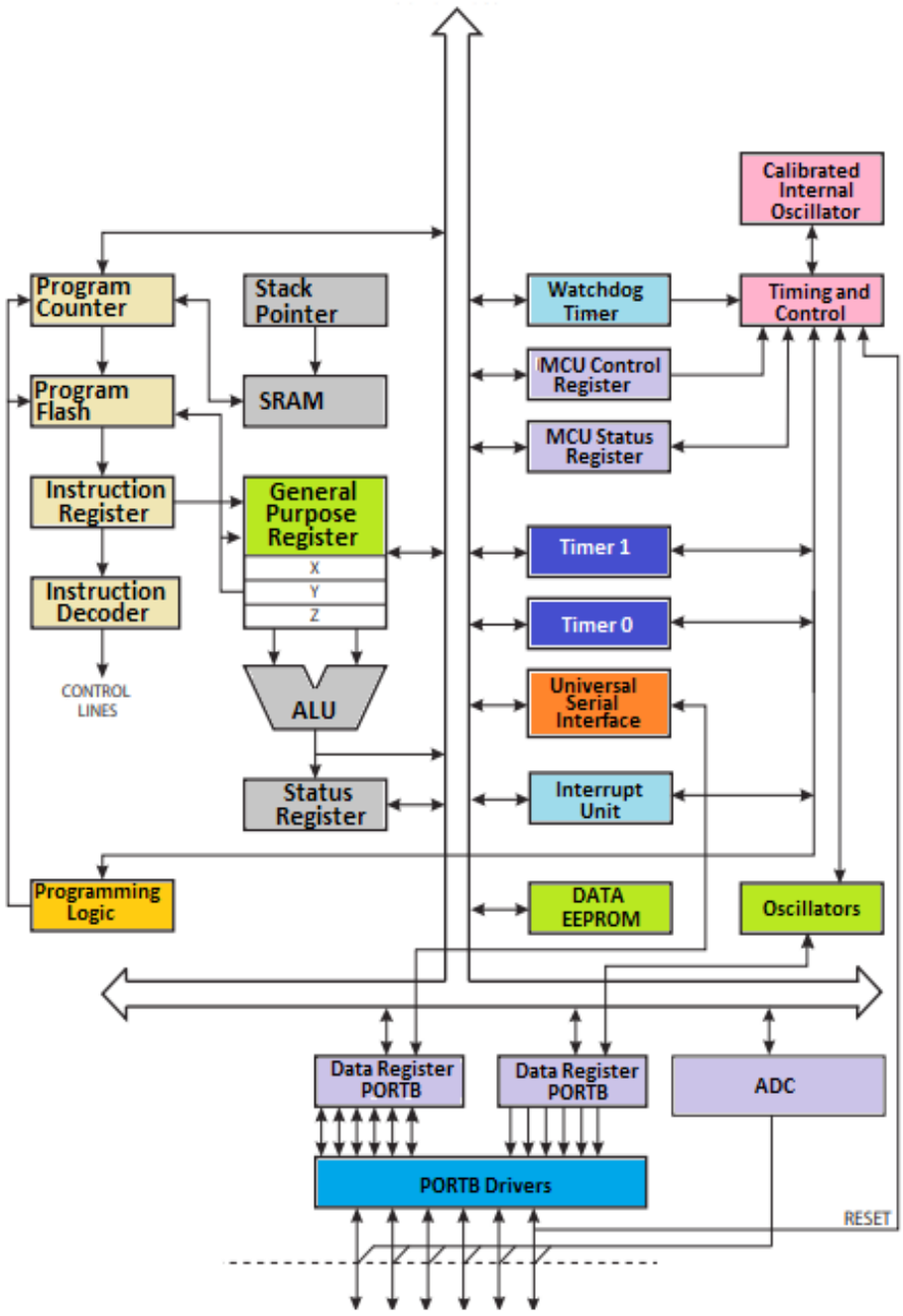

Рисунок 1.3 – Структура мікроконтролера ATtiny85

Ядро AVR комбінує багатий набір інструкцій з 32 регістрами загального призначення, які безпосередньо підключені до арифметико-логічного пристрою. Це дозволяє здійснювати доступ при виконанні інструкції відразу до двох регістрів і виконати її за один машинний цикл. Результуюча архітектура має більш високу ефективність, забезпечуючи продуктивність в більш ніж 10 разів вище в порівнянні з традиційними CISC-мікроконтролерами [1, 5].

1.2 Програматор

Програматор – пристрій призначений для запису інформації у постійний запам'ятовуючий пристрій. В даній кваліфікаційній роботі він потрібен для загрузки коду програми в постійну пам’ять мікроконтролера ATtiny85.

Крім запису, програматор може забезпечувати можливість зчитування інформації з постійної пам’яті мікросхеми. Поряд з основними режимами програмування і читання, багато мікросхеми мають безліч додаткових режимів, таких як:

- стирання;
- контроль частоти;
- програмування;
- верифікація;
- конфігурація.

Для виконання даної атестаційної роботи було обрано один з найпоширеніших та простих у використанні програматорів – USBASP v.2.0.

Програматор USBASP дозволить інженеру отримати простий, компактний і надійний програматор всіх мікроконтролерів сімейства AVR з режимом послідовного програмування ISP (In System Programming). Це зручний і мініатюрний програматор, що підключається до USB-порту персонального комп'ютера, що дуже актуально, через те що COM-порт існує далеко не на всіх сучасних комп'ютерах, і тим більше на ноутбуках.

Використання USB програматора USBASP і його функції внутрісистемного програмування дають можливість швидко і багаторазово програмувати мікроконтроллерні пристрій в зібраному вигляді, не відключаючи його від живлення. При цьому процес налагодження програмного забезпечення за допомогою даного AVR програматора USBASP помітно спрощується і скорочується витрачається на цей час. Також варто зазначити, що даний програматор сумісний з усіма сучасними операційними системами, такими як: Linux, MacOS та Windows. Нижче на рисунку 1.4 наведено зображення данного програматора.

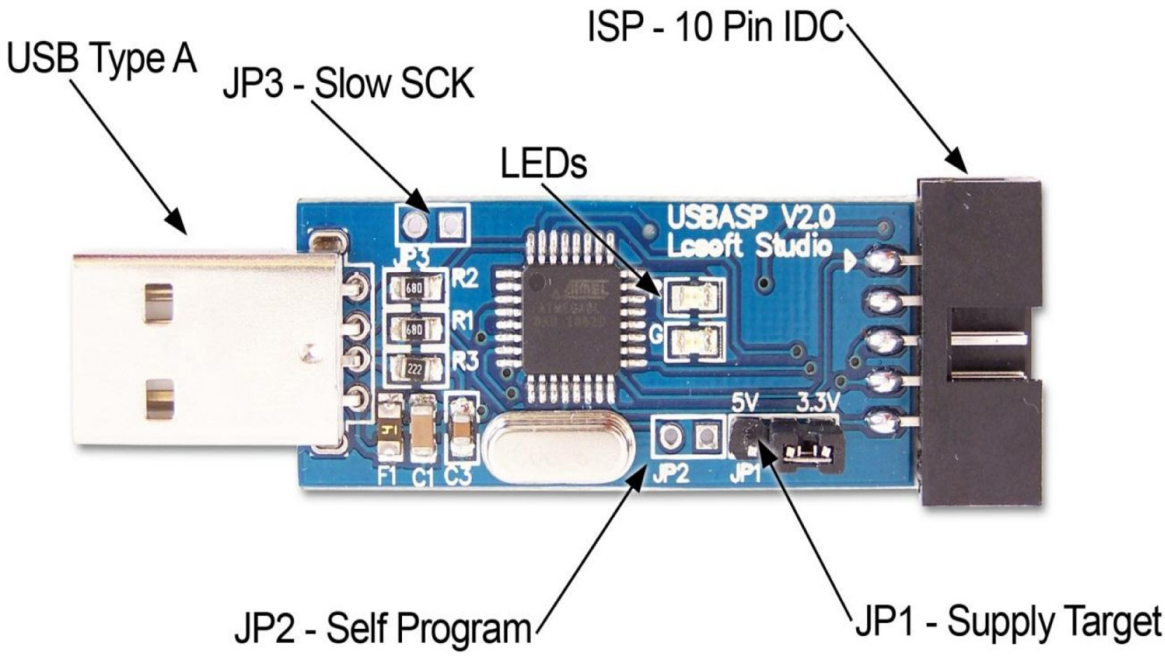

Рисунок 1.4 – Зображення USBASP програматора

На рисунку 1.4 вказані основні апаратні частини програматора, такі як:

– порт USB Type A – USB-кінець програматора, котрий підключається безпосередньо до USB порту комп’ютера;

– порт послідовного програмування ISP;

– LED світлодіоди – На корпусі USBASP програматора розташовано два

світлодіода за маркуваннями R та G відповідно. Світлодіод R використовується для індикації обміну даними з комп'ютером (запис або зчитування даних). Світлодіод G сигналізує про те, що програматор знаходиться в робочому стані;

– роз'єм JP1 - призначений для запису коду програми в мікроконтролер самого програматору;

– роз'єм JP2 - напруга живлення програматора - 5 Вольт або 3,3 Вольта (за замовчуванням програматор встановлено в режим роботи 5 Вольт);

– роз'єм JP3 - визначає частоту запису даних SCK. В розімкнутому стані висока частота (375 кГц), у замкнутому низька частота (8 кГц).

Обмін даними між мікроконтролером та програматором забезпечує десятиконтактний роз’єм ISP (режим послідовного програмування). На рисунку 1.5 наведено призначення контактів ISP роз’єму.

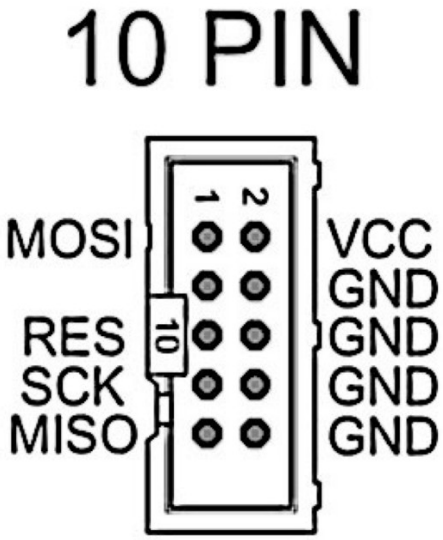

Рисунок 1.5 – Схема контактів ISP роз’єму

Нижче наведено призначення усіх контактів зображених на рисунку 1.5:

- MOSI – вихід даних для послідовного програмування;
- VCC +5V – вихід + 5В, для живлення зовнішнього пристрою від шини;
- NC – ве використовується;

- GROUND – загальний або мінус живлення;
- RESET – підключається до RESET виходу мікроконтролерв;
- GROUND – загальний або мінус живлення;
- SCK – вихід тактування даних;
- GROUND – загальний або мінус живлення;
- MISO – вихід данних для послідовного програмування;
- GROUND – загальний або мінус живлення.

1.3 Постановка задачі

Предметом даного дослідження виступає технологія RFID, її специфікації та стандарти. У якості об’єктас служить універсальний цифровий RFID ключ.

Основною метою даної атестаційної роботи є розробка універсального цифрового ключа для систем контролю та управління доступом на основі технології RFID.

Для досягнення поставленої мети мають бути вирішені настепні задачі:

- аналіз математичного апарату та специфікацій технології RFID;
- проектування та реалізація апаратної частини проекту;
- проектування та реалізація програмної частини проекту;
- аналіз отриманих результатів.

2 ОГЛЯД ІСНУЮЧИХ АПАРАТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

2.1 Мікроконтролер

Згідно з поставленою задачею метою даної атестаційної роботи блула реалізація комп'ютерної системи та розробка програмного алгоритма для ефективного відображення графічних зображень та анімацій. Для виконання поставленого завдання, треба виділити основні вимоги до обраного мікроконтролера:

– компактність готової системи;

– висока продуктивність системи та можливість проводити десятки тисяч операцій в секунду;

– невисока ціна готової системи;

– наявність достатнього об'єму енергонезалежної пам'яті для демонстрації можливостей оптимізації зберігання та відображення графічних зображень;

– енергоефективність готової системи [4].

Згідно з перерахованими вище вимогами в якості головного компонента комп'ютерної системи було обрано мікроконтролер ATtiny85. Поступово розглянемо чи підходить він піп вказані вище параметри.

ATtiny85 – молодший у лінійці мікроконтролерів AVR та має найкомпактніший корпус та меншу собівартість. Мікроконтролер має менше число ліній введення-виведення, що не е критичним для даної атестаційної роботи через відсутність потреб в підключенні великої кількості периферійних пристроїв. Також варто зазначити, що ATtiny85 має таку ж саму продуктивність як і старші мікроконтролери сімейства mega [2, 5]. Як ми бачимо, мікроконтролер ATtiny85 – чудовий вибір мікроконтролера для атестаційної роботи, який задовольняє усі поставлені умови.

2.2 Дісплейний модуль

Для виведеня графічної інформації та зображень користувачеві було обрано однокольоровий дісплейний модуль. Нижче на рисунку 4.1 наведено зображення обраного дісплейного модуля.

Рисунок 2.1 – Зображення дісплейного модуля

Дисплей діагоналлю 0,96" виконаний за технологією OLED, завдяки чому, він не потребує підсвічуванні на відміну від РК дисплеїв. OLED-0.96-128X64 виконаний з 128x64 окремих OLED пікселів, кожен з яких управляється за допомогою мікросхеми контролера. Дисплей підтримує I2C інтерфейс.

I2C – послідовна шина даних для зв'язку інтегральних схем, розроблена фірмою Philips на початку 1980-х як проста шина внутрішнього зв'язку для створення керуючої електроніки. Використовується для з'єднання низькошвидкісних периферійних компонентів з материнською платою, вбудовуваними системами та мобільними телефонами. Назва є абревіатурою слів Inter-Integrated Circuit.

Нижче наведено характеристики графічного дісплея, зображеного на рисунку 2.1:

- щільність екрану: 128x64;
- діагональ: 0,96";
- кут огляду: 160 °;
- колір свічення: синій;
- інтерфейс: I2C;
- тип драйвера матриці: SSD1306;
- робоча температура: -30 ... 70 ° C.;
- напруга живлення: 3 ... 5 В.;
- драйвер: SSD1306.;
- сумісний з середовищем Arduino, MSP430, STM32;
- розміри: 27x27x4,1 мм [1, 6].

2.3 Середовище програмування

Для виконання даної атестаційної роботи було обрано середу програмування Arduino IDE, найпопулярнішу платформу аматорської та професійної електроніки та робототехніки. Нижче розглянемо її найголовніщі плюси.

Просте і зрозуміле середовище програмування Arduino IDE підходить як для початківців, так і для досвідчених користувачів. Arduino IDE заснована на середовищі програмування Processing, що дуже зручно для викладачів, так як студенти працюють з цим середовищем будуть знайомі і з Arduino.

Можливість живлення, програмування та обміну повідомленнями з програмованним мікроконтролером за допомогою одного USB кабелю (або FTDI кабелю для деяких мікроконтролерів).

Наявність великої кількості бібліотек для широкого спектру задач від зчитування сигналів кнопок, виведення інформації на семисегментні або ЖК-дисплеї до управління двигунами, для всіх задач є стандартні бібліотеки, які не

потребують великого досвіду в програмуванні.

Проработана механика для роботи з послідовними і SPI інтерфейсами.

Середовище розробки Arduino IDE складається з вбудованого текстового редактора програмного коду, області повідомлень, вікна виведення тексту чи консолі, панелі інструментів з кнопками часто використовуваних команд і декількох меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини.

Програма, написана в середовищі Arduino IDE, називається скетч. Скетч пишеться в текстовому редакторі, що має інструменти вирізки / вставки, пошуку / заміни тексту. Під час збереження і експорту проекту в області повідомлень з'являються пояснення, також можуть відображатися виникли помилки. Вікно виведення тексту чи консоль показує повідомлення Arduino, що включають повні звіти про помилки та іншу інформацію. Кнопки панелі інструментів дозволяють перевірити і записати програму, створити, відкрити і зберегти скетч, відкрити моніторинг послідовної шини.

Після запуску програми ви можете знайти чотири головних функціональних елементи:

– меню – містить усі опції програми;

– панель інструментів – містить найбільш часто використовувані функції програми;

– вікно з кодом програми – текстовий редактор для написання коду програми (скетчу);

– інформація програми – в цьому вікні виводитися інформація про помилки в коді та який обсяг пам'яті зайняла програма в пам'яті програмованого контролера;

– номер поточного рядка – номер рядка, на якому перебуває курсор;

– закладка редактора – дозволяє записувати одну програму в декількох вкладах / файлів. Це дозволяє розділити велику програму на менші простіші в розумінні шматки.

2.4 Мова програмування

Написання коду для обраного мікроконтролера буде здійснюватись мовою програмування С. Мова програмування С – найшвидша у світі високорівнева мова програмування. Для мови С характерні лаконічність, стандартний набір конструкцій управління потоком виконання, структур даних і великий набір операцій. Програма, написана на С для однієї обчислювальної системи, може бути перенесена з невеликими змінами (або взагалі без них) на іншу. Якщо модифікації все-таки необхідні, то часто вони можуть бути зроблені шляхом простої зміни декількох елементів в "головному" файлі [3].

С – універсальна, процедурна, імперативна мова програмування загального призначення, розроблена у 1972 році Деннісом Рітчі у Bell Telephone Laboratories з метою написання нею операційної системи UNIX. Хоча С і було розроблено для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення. Серед її головних цілей: можливість прямолінійної реалізації компіляції, використовуючи відносно простий компілятор, забезпечити низькорівневий доступ до оперативної пам'яті, формувати лише кілька інструкцій машинної мови для кожного елементу мови і не вимагати великої динамічної підтримки. У результаті код С придатний для більшості системного програмного забезпечення, яке традиційно писали асемблером. С особливо популярний у системних програмістів, тому що дозволяє писати програми просто та коротко

Незважаючи на її низькорівневі можливості, мову проектували для машинно-незалежного програмування. Сумісна зі стандартами та машинно-незалежно написана мовою С програма може легко компілюватися на великій кількості апаратних платформ та операційних систем з мінімальними змінами. Мова стала доступною для великої кількості платформ - від вбудованих

мікроконтролерів до суперкомп’ютерів [3].

Довгий цикл життя даної мови програмування можна пояснити тим, що принципи роботи операційних систем відносно універсальні, вони не схильні до того прогресу та різноманітності, які можна спостерігати в середовищі десктопного та мобільного ПЗ, Web-додатків. С не є мовою досить високого рівня, вона ближче до архітектури комп'ютера. В результаті програми на С виходять швидше.

Сі у чистому вигляді не підтримує об'єктно-орієнтованого програмування (хоча є бібліотека, де емулюються можливості ООП). Підтримка ООП реалізована C++. Хоча останній виник на основі мови С, він не є його "продовженням", а є окремою мовою, яку можна вивчати, не знаючи С. Проте вивчення С корисне перед знайомством з його "просунутим молодшим братом", т.к. синтаксис мов схожий, С не перевантажує мозок програміста-початківця надможливостями і привчає до розуміння суті того, що відбувається.

На прикладі найпростішої програми одразу відзначимо деякі особливості мови програмування С.

У мові С роль основної гілки програми бере функція main(). Вона є точкою входу до програми. Ця функція завжди повинна бути присутня в закінченій програмі мовою С, і виконання програми починається саме з неї. Проте оголошені всередині неї змінні є глобальними, їх область видимості простягається лише на функцію main(). Проте в мові програмування С майже весь програмний код полягає у функції, і функція main() є головною та обов'язковою.

За замовчуванням функція main() повертає тип даних int, тому можна не вказувати тип даних, що повертаються. Проте компілятор у разі виносить попередження.

Функція printf() призначена виведення даних. Її призначення аналогічне функції print() у Python. Однак функція printf() після виведення не виконує перехід на новий рядок. Тому для переходу використовується спеціальний

символ, що позначається комбінацією \n. Вирази мовою С поділяються крапкою з комою.

У мові С функції введення-виводу є частиною мови. Наприклад, у Python нам не треба імпортувати жодний модуль, щоб користуватися функціями print() та input(). У С ми не можемо просто викликати функцію printf(), тому що в самому С її просто немає. Цю функцію, а також ряд інших, можна підключити за допомогою файлу stdio.h. Саме для цього на початку програми прописан рядок #include <stdio.h>. Include з англійської перекладається як "включити", а stdio є скорочення від "стандартне введення-виведення (input-output)".

У заголовних файлах (вони закінчуються на .h) зазвичай містяться оголошення тих чи інших функцій. Оголошення – це просто опис функції: які параметри вона приймає та що повертає. Сам код функції (визначення) знаходиться не в заголовному файлі, а в бібліотеках (інших файлах), які можуть бути скомпільовані і розташовані в системних каталогах. Перед компіляцією програми запускається препроцесор мови С. Крім іншого він включає початок файлу програми вміст вказаних у ній заголовних файлів.

Сі має два основних типи даних (змінних): ціле і символьне значення, що оголошуються як int і char, відповідно. Немає окремої булевої змінної. Як булева змінна використовується змінна int. Якщо ця змінна містить 0, це означає брехня/false, а будь-яке інше значення означає істина/true. Сі також має і типи з плаваючою точкою, але MINIX не використовує їх.

До типу int можна застосовувати «прикметники» short, long або unsigned, які визначають діапазон значень, що залежить від компілятора. Більшість процесорів 8088 використовують цілі 16-бітні числа для int і short int і 32-бітові цілі числа для long int. Цілі числа без знака (unsigned int) на процесорі 8088 мають діапазон від 0 до 65535, а не від -32768 до +32767, як у звичайних цілих чисел (int). Символ займає 8 біт.

Специфікатор register також допускається як для int, так і для char, і є підказкою для компілятора, що оголошену змінну варто помістити в регістр,

щоб програма працювала швидше.

Перетворення між типами дозволено. В багатьох випадках необхідно чи корисно примусово проводити перетворення між типами даних. Для примусового перетворення достатньо поставити цільовий тип у дужках перед виразом перетворення. У разі перетворення між типами слід звернути увагу на знак.

При перетворенні символу на ціле число деякі компілятори обробляють символи як знакові, тобто від - 128 до +127, тоді як інші розглядають їх як без знака, тобто від 0 до 255.

3 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ RFID

3.1 Специфікації стандарту RFID

Обсяг потоку товарів збільшується по всьому світу, зростаючі вимоги до функції відстеження і необхідність зростання ефективності виробничих процесів роблять необхідною таку інформаційну систему, за допомогою якої можна чітко і достовірно ідентифікувати товар або упаковку і супроводжувати їх конкретними даними. Радіочастотна ідентифікація надає таку можливість. В результаті все більше компаній довіряє цій технології, яка знаходить застосування в мільярдах областей, у тому числі в протиугінних пристроях автомобілів, в проїзних документах, вхідних квитках на стадіони, а також в системах контролю доступу в будівлі. Всі ці програми об'єднує те, що вони підтримуються універсальною концепцією використання різних компонентів апаратного забезпечення та програмного забезпечення.

RFID або радіочастотна ідентифікація – це спосіб автоматичної ідентифікації об'єктів, в якому за допомогою радіосигналів зчитуються або записуються дані, що зберігаються в так званих транспонтерах, або RFID-мітках.

Система радіочастотної ідентифікації RFID включає до свого складу обладнання та програмне забезпечення:

- RFID-мітка складається з мікрочіпа, який зберігає інформацію для ідентифікації, і антени, за допомогою якої прилад передає і отримує дані;
- RFID-зчитувачі (рідери, сканери) розпізнають радіопозначки, отримують відомості і відправляють їх в базу даних;
- RFID-антени випромінюють електромагнітні сигнали, які активізують мітки для вільного читання / запису інформації;

– програмне забезпечення складається з серверної частини і клієнтських додатків. Забезпечує взаємодію автоматизованих інформаційних систем. (АІС) з РЧІД системою, а також віддалене адміністрування обладнанням.

Технологія RFID з моменту свого створення здобула вже більше десятка різних стандартів. Стандарти радіочастот спочатку були прийняті Міжнародною Організацією Стандартизації для діапазону LF (Low Frequency – діапазон з низькою робочою частотою) частоти в 125 кГц. Найбільш популярними вважалися ISO 11784 та ISO 11785 застосовувані в тваринництві, але згодом кількість стандартів, починаючи з 1996 року значно збільшилася.

Таблиця 3.1 – Основні стандарти технології RFID

Робоча частота	**Стандарт**	**Призначення**
125-135 КГц	• ISO 14223 • ISO 11784 • ISO 11785 • ISO 18000-2	Розроблявся і застосовувався для ідентифікації тварин в сільському господарстві, а й зараз використовуються досить широко, наприклад, в автомобільних системах безпеки та СКУД
13,56 МГц	• ISO 14443 • ISO 15693 • ISO 10373 • ISO 18000-3	Побутових домофонах і ритейлі. Найчастіше представлені у вигляді безконтактних смарт-карт. Працює на близьких відстанях до декількох сантиметрів. Дана частота лежить в основі технології NFC
860-930 МГц	• ISO 15961 • ISO 15962 • ISO 15963 • ISO 18000-6	Найбільш поширений діапазон в застосуванні RFID технології. В основному задіяний в роботі з логістичними операціями і транспортуванням. Варто виділити окремий напрямок складського обліку із середньою дальністю ідентифікації.
2,45 ГГц	• ISO 15961 • ISO 15962 • ISO 15963 • ISO 18000-4	Застосовується в системах з зоною реєстрації на збільшені відстані. При проектуванні вимагає врахування безлічі факторів, що впливають на

		якість сигналу

3.2 Технологія RFID в системах контролю та управління доступом

Технологія RFID вже давно не є тільки академічного інтересу. Спектр застосування її в світі поширюється. Тематичні видання рясніють повідомлення про нові проекти, виконаних за допомогою радіочастотної ідентифікації. Розглянемо поширені рішеннях в різних галузях: в промисловості, в складському господарстві, індустрії розваг, транспорті та іншому.

Для будь-якої системи безпеки є два важливих чинників: пропускна здатність і надійність роботи. Спробуємо оцінити, чому технологія RFID може бути тут надзвичайно корисною.

Зазвичай для задач контролю доступу застосовують безконтактні RFID-карти і компактні зчитувачі для них. За своїми розмірами вони точно відповідають стандарту ISO 7816 (описує пристрій ідентифікаційних карт), легкі, можуть бути привабливо оформлені і мають наступні переваги: високий ступінь захисту даних в пам'яті, включаючи унікальний ID-номер, що формується при виготовленні мікросхеми і який неможливо модифікувати, і безконтактний принцип роботи.

Перше дає гарантії безпеки і ускладнює підробку електронного посвідчення, друга ж забезпечує тривалий термін служби, оскільки при безконтактному зчитуванні механічних пошкоджень або забруднення контактної площадки можна не побоюватися, як це могло б бути зі смарт-картами. У комплекті з RFID системою контролю доступу та обліку робочого часу йде програма для бухгалтерії, що дозволить істотно скоротити час впровадження системи обліку робочого часу персоналу.

Апаратура досить легко інтегрується в існуючу інформаційну середу завдяки вибору інтерфейсів підключення: RS-232, 485, TTL, Wiegand, іноді навіть Ethernet. Це дозволяє пов'язати точки контролю доступу на місцях з центром контролю доступу - спеціальним серверним програмним

забезпеченням, яке приймає рішення надати доступ (відкрити двері, турнікет і т.п.) або відмовити згідно прив'язці даних мітки до прав доступу в базі даних співробітників.

Система контролю доступу на основі безконтактних карт здатна вирішувати весь комплекс завдань по автоматизації охорони території в цілому і приміщень з обмеженим доступом.

При виборі спеціалізованих автотранспортних СКУД споживачам варто звертати увагу на такі моменти.

Слід перевіряти наявність типових інтерфейсів для підключення до контролерів СКУД. Це особливо актуально при інтеграції автотранспортної та класичної СКУД (облік людей). Необхідно також уточнювати, як зчитувач видає код в контролер СКУД.

Вибрані ідентифікатори повинні бути ефективними та зручними, вони повинні відповідати вашим вимогам щодо дистанції читання, розмірів, міцності, кліматичних умов, дизайну і тому подібне.

При інтеграції транспортної та класичної СКУД іноді можна знизити витрати, якщо використовувати RFID-системи подвійного призначення або системи з мітками, що містять відразу два ідентифікатори, низько та високочастотні ідентифікатори від різних виробників.

Бажано, щоб обрані зчитувачі були зручними для монтажу та експлуатації. Навряд чи буде зручно лізти на стовп із комп'ютером у руках, щоб перепрограмувати зчитувач. Також буде дуже сумно, якщо він вийде з ладу через кліматичні особливості місця, де він експлуатується.

Оскільки йдеться про ідентифікацію автотранспорту, слід враховувати, на якій швидкості здійснюватиметься реєстрація автомобільної мітки, оскільки цей параметр може відрізнятися у різних виробників у декілька разів.

Нижче, на рисунку 3.1, наведена загальна схема конфігурації системи контролю та управління доступом функціонуючій на основі технології RFID [7].

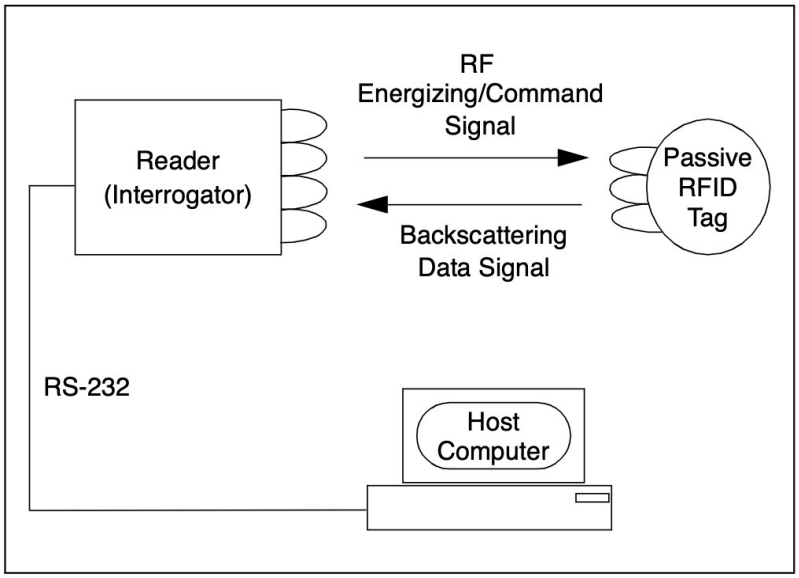

Рисунок 3.1 – Схема СКУД за технологією RFID

Побудова системи контроля та управляння доступом навколо технології RFID надає системі ряд переваг, які будуть перелічені нижче:

- RFID не потрібує контакт або пряма видимість;
- RFID-мітки читаються швидко і точно;
- RFID може використовуватися навіть в агресивних середовищах, а RFID-мітки можуть читатися через бруд, фарбу, пар, воду, пластмасу, деревину;
- пасивні RFID-мітки мають фактично необмежений термін експлуатації;
- RFID-мітки несуть велику кількість інформації і можуть бути інтелектуальними;
- RFID-мітки можуть бути не тільки для читання, але і з записом інформації.

Апаратура, що зчитує, досить легко інтегрується в існуюче інформаційне середовище завдяки вибору інтерфейсів підключення: RS-232, 485, TTL, Wiegand, іноді навіть Ethernet. Це дозволяє ув'язати точки контролю доступу на місцях з центром контролю доступу – спеціальним серверним програмним забезпеченням, яке приймає рішення надати доступ (відчинити двері, турнікет

тощо) або відмовити згідно з прив'язкою даних мітки до прав доступу в базі даних співробітників.

Багаторівнева система контролю доступу на основі безконтактних карток здатна вирішувати весь комплекс завдань з автоматизації охорони території в цілому та приміщень з обмеженим доступом.

Крім власне пропускної системи, персональні карти співробітників можуть використовуватися для керування низкою додаткових функцій: харчуванням співробітників, обліку робочого часу тощо.

3.3 Критерії обрання RFID системи

В останні роки відбулося величезне зростання популярності систем RFID. Найкращим прикладом цього явища є безконтактні смарт-картки, які використовуються як електронні квитки на громадський транспорт. П’ять років тому було неможливо уявити, що тепер використовуються десятки мільйонів безконтактних квитків. Можливі сфери застосування систем безконтактної ідентифікації останнім часом також розширилися.

Розробники систем RFID врахували цю розробку, в результаті чого зараз на ринку доступна незліченна кількість систем.

Технічні параметри цих систем оптимізовані для різних сфер застосування – оформлення квитків, ідентифікація тварин, промислова автоматизація або контроль доступу. Технічні вимоги цих сфер застосування часто збігаються, а це означає, що чітка класифікація відповідних систем не є простою справою. Ситуацію ще більш ускладнює те, що за винятком кількох особливих випадків (ідентифікація тварин, смарт-карти тісного зв’язку), для систем RFID ще не існує жодних обов’язкових стандартів.

Навіть фахівцеві важко зберегти уявлення про асортимент систем RFID, які зараз пропонуються. Тому користувачам не завжди легко вибрати систему, яка найкраще відповідає їхнім потребам.

3.3.1 Робоча частота

Системи RFID, які використовують частоти приблизно від 100 кГц до 30 МГц, працюють за допомогою індуктивного зв’язку. Мікрохвильові системи навпаки в діапазоні частот 2,45–5,8 ГГц підключаються за допомогою електромагнітних полів. Питома швидкість поглинання (згасання) для води або непровідних речовин нижча в 100 000 разів на частоті 100 кГц, ніж при 1 ГГц. Тому практично не відбувається поглинання або демпфування. Низькочастотні радіочастотні системи в першу чергу використовуються завдяки кращому проникненню об’єктів. Прикладом цього є спеціально розроблені мітки для великої худоби, які завдяки ін’єкції вживлюються тварині та які можна зчитувати ззовні на частоті запиту приблизно у 135 кГц.

Мікрохвильові системи мають значно більший діапазон, ніж індуктивні, зазвичай 2–15 м. Однак, на відміну від індуктивних систем, мікрохвильові системи потребують додаткової резервної батареї. Потужності передачі зчитувача, як правило, недостатньо для забезпечення достатньої потужності для роботи транспондера.

Іншим важливим фактором є чутливість до полів електромагнітних перешкод, таких як ті, які генеруються зварювальними роботами або потужними електродвигунами.

Індуктивні транспондери тут мають суттєвий недолік. Тому мікрохвильові системи особливо зарекомендували себе на виробничих лініях і фарбувальних системах автомобільної промисловості. Іншими факторами є велика ємність пам'яті (до 32 Кбайт) і стійкість до високої температури (до 250 градусів за цельсієм) мікрохвильових систем.

3.3.2 Відстань

Необхідний діапазон застосування залежить від кількох факторів:

– точність позиції транспондера;
– мінімальна відстань між декількома транспондерами в практичній експлуатації;
– швидкість транспондера в зоні опитування зчитувача.

Наприклад, у програмах безконтактних платежів, наприклад квитки на громадський транспорт – швидкість позиціонування дуже низька, оскільки транспондер наводиться до зчитувача вручну кожним окремим користувачем системи.

Мінімальна відстань між декількома транспондерами в цьому випадку відповідає відстані між двома пасажирами, що в'їжджають у транспортний засіб. Для таких систем є оптимальний діапазон 5–10 см. Більший діапазон у цьому випадку лише спричинить проблеми, оскільки читач може виявити одночасно кілька квитків пасажирів. Це унеможливить надійне розподіл квитка відповідному пасажиру.

На виробничих лініях автомобільної промисловості часто одночасно виготовляють різні моделі транспортних засобів різних розмірів. Таким чином, великі варіації відстані між транспондером на транспортному засобі та зчитувачем заздалегідь запрограмовані. Тому відстань запису/зчитування використовуваної системи RFID має бути розроблена для максимально необхідного діапазону. Відстань між транспондерами має бути такою, щоб у зоні опитування зчитувача одночасно знаходився лише один транспондер. У цій ситуації мікрохвильові системи, в яких поле має спрямований промінь, мають очевидні переваги перед широкими ненаправленими полями систем з індуктивним зв’язком.

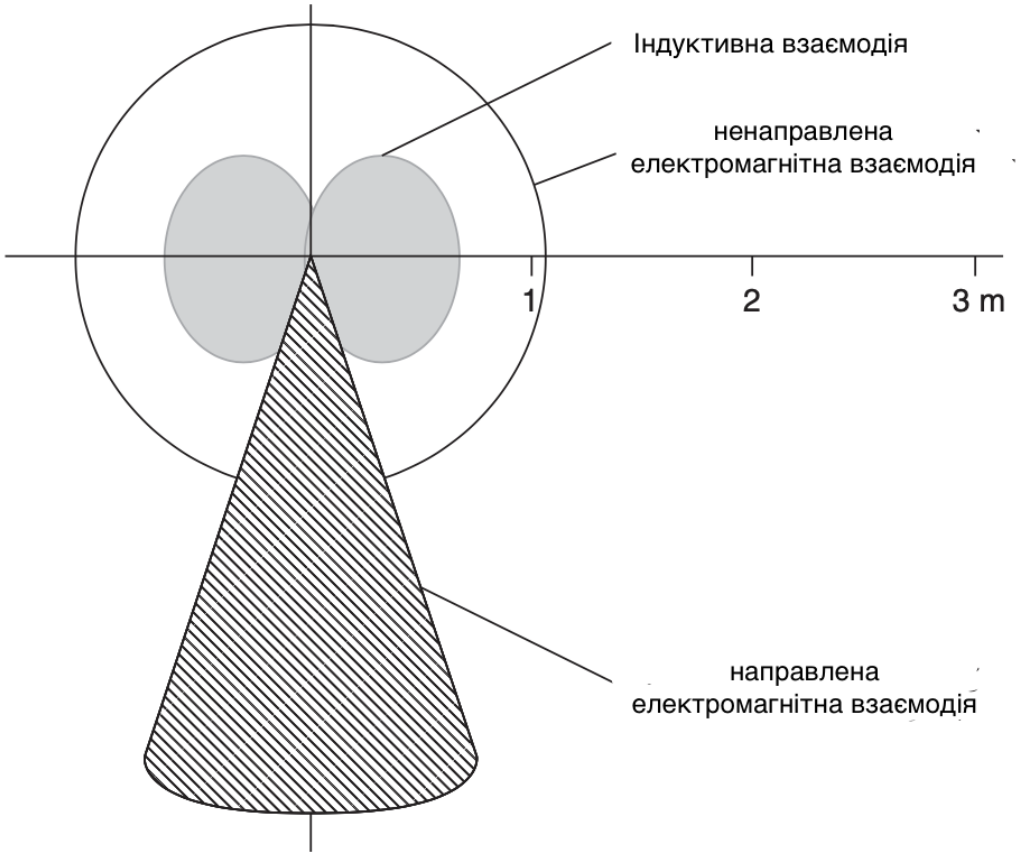

Рисунок 3.2 – Порівняння відносних зон взаємодії різних систем

Швидкість транспондерів по відношенню до зчитувачів разом з максимальною відстанню запису/читання визначає тривалість часу, проведеного в зоні опитування читача. Для ідентифікації транспортних засобів необхідна дальність дії системи RFID розроблена таким чином, щоб при максимальній швидкості транспортного засобу тривалість перебування в зоні опитування була достатньою для передачі необхідних даних [7].

3.3.3 Вимоги до безпеки системи

Вимоги до безпеки, які будуть накладені на запланований додаток RFID, тобто шифрування та аутентифікація, повинні бути дуже точно оцінені, щоб виключити будь-які неприємні сюрпризи на етапі впровадження. Для цього слід

оцінити стимул, який система представляє потенційному зловмиснику як засіб придбання грошей або матеріальних благ шляхом маніпуляцій. Щоб мати можливість оцінити цю привабливість, ми поділяємо заявки на дві групи:

– промислові або закриті програми;
– публічні програми, пов'язані з грошима та матеріальними благами.

Це можна проілюструвати на основі двох контрастних прикладів застосування. Давайте ще раз розглянемо складальну лінію в автомобільній промисловості як типовий приклад промислового або закритого застосування. Доступ до цієї системи RFID мають лише уповноважені особи, тому коло потенційних зловмисників залишається досить невеликим.

Зловмисна атака на систему шляхом зміни або фальсифікації даних аутентифікації на транспондері може призвести до критичної несправності в операційній послідовності, але зловмисник не отримає жодної особистої вигоди. Таким чином, ймовірність атаки може бути встановлена рівною нулю, що означає, що можна використовувати навіть дешеву недорогу систему без логіки безпеки.

Наш другий приклад — це система продажу квитків для використання в громадському транспорті. Така система, насамперед носії даних у вигляді безконтактних смарт-карт, доступна кожному. Таким чином, коло потенційних нападників величезне.

Успішна атака на таку систему може представляти собою масштабну фінансову шкоду відповідній компанії громадського транспорту, наприклад, у разі організованого продажу фальсифікованих проїзних проїзних, не кажучи вже про шкоду іміджу компанії. Для таких додатків незамінний транспондер високого класу з процедурами аутентифікації та шифрування. Нехтування вимогами безпеки можуть призвести до серьйозних фінансових наслідків.

Для програм з максимальними вимогами безпеки, наприклад банківських програм з електронним гаманцем, слід використовувати тільки транспондери з мікропроцесорами.

3.3.4 Ємність пам'яті

Розмір мікросхеми носія даних, отже і ціновий клас – в першу чергу визначаються його обсягом пам’яті. Таким чином, постійно закодовані носії даних лише для читання використовуються в чутливих до ціни масових додатках з низькими вимогами до локальної інформації. Однак за допомогою такого носія даних можна визначити лише ідентичність об’єкта. Подальші дані зберігаються в центральній базі даних керуючого комп'ютера. Якщо дані мають бути записані назад на транспондер, потрібен транспондер з технологією EEPROM або RAM.

Пам'ять EEPROM в основному зустрічається в системах з індуктивним зв'язком. Є пам’ять від 16 до 8 Кбайт. З іншого боку, пристрої пам’яті SRAM з резервним акумулятором використовуються переважно в мікрохвильових системах. Об’єм пам’яті, який пропонується, коливається від 256 байт до 64 Кбайт.

4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ

4.1 Проектування та реалізація апаратної частини проекту

Апаратна частина цифрового ключа складатиметься з мікроконтролера Attiny85 як центрального компоненту системи, дісплейного модулю для відображення інтерфейсу користувачеві, дві контактні кнопки в якості джерела вхідного сигналу та котушки індуктивності на 125 kHz згідно з стандартом ISO 14223, технології RFID. Цей стандарт вважається сучасним та широко використовується в автомобільних системах безпеки та системах контролю та управління доступом.

У якості джерела живлення для дісплейного модуля та мікроконтролера було обрано батарейку типу CR2032 з напругою живлення у 3В котра широко використовується у наручних годинниках, пультах дистанційного керування, фонариках, брелках та в інших приладах.

Для того, щоб дісплейний модуль був спроможній виводити графічну інформацію, треба послідовно з’єднати виходи мікроконтролера з виходами дісплейного модуля. Схема розташування виводів мікроконтролера ATtiny85 наведено на рисунку 1.2. Завдяки цій схемі ми послідовно з’єднаємо виходи мікроконтролера ATtiny85 з виходами дісплейного модуля (рисунок 3.2):

Також треба з’єднати плюсовий вивід джерела живлення з VCC виводом мікроконтролера ATtiny85 та мінусовий вивід з GND виводом мікроконтролера. Також заради простоти використання і зручності в процесі розробки до схеми можно додати перемикач живлення.

У якості джерела вхідного сигналу в даній системі виступає контактна кнопка (рисунок 4.1). Її під’єднано через резистор номіналом 200 Ом до другого входа мікроконтролера Attiny85.

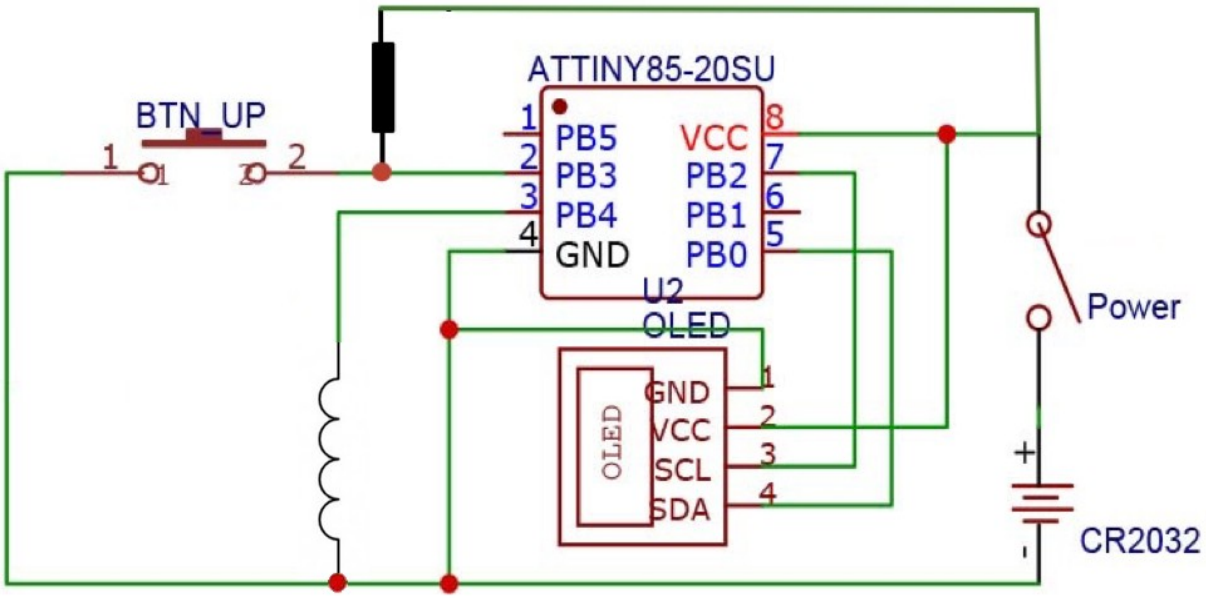

Рисунок 4.1 – Схема підключення компонентів

У якості джерела вхідного сигналу в системі виступає котушка індуктивності на 125 kHz, котру під’єднано до третього входу мікроконтролера. Типова кількість витків котушки знаходиться в діапазоні 100 витків для 125 кГц і 3~5 витків для пристроїв 13,56 МГц. Для більшого діапазону зчитування котушка антени повинна бути належним чином налаштована на цікаву частоту.

Електричний струм, що протікає через провідник, створює магнітне поле. Це змінюване у часі магнітне поле здатне виробляти потік струму через інший провідник.

Це змінюване у часі магнітне поле і називається індуктивністю. Індуктивність L залежить від фізичних характеристик провідника. Котушка має більшу індуктивність, ніж прямий дріт з того ж матеріалу, а котушка з більшою кількістю витків має більшу індуктивність, ніж котушка з меншою кількістю витків.

Індуктивність L котушки індуктивності визначається як відношення загального магнітного потоку до струму I через індуктивність (формула 4.1).

$$L = \frac{N\psi}{I} \tag{4.1}$$

Де, N – кількість витків у котушці;

Ф – Магнітний потік;

I – сила струму.

У типовій котушці антени RFID для 125 кГц індуктивність часто вибирається як мГн для мітки і мкГн для зчитувача. Для котушки антени з кількома витками, більша індуктивність призводить до більш плотного розташування витків. Тому котушка для RFID-мітки, яка має бути сформована в обмеженому корпусі, часто потребує багатошарової обмотки, щоб зменшити кількість витків. Конструкція індуктора здається відносно простою справою. Однак побудувати ідеальний індуктор практично неможливо, оскільки котушка має кінцеву провідність, що призводить до втрат, і розподілена ємність існує між витками котушки та між провідником та навколишніми предметами.

Фактична індуктивність завжди є комбінацією опору, індуктивності та ємності. Уявна індуктивність — це ефективна індуктивність на будь-якій частоті, тобто індуктивна мінус ємнісний ефект. Нижче на рисунку 4.2 наведено зображення примітивної котушки індуктивності.

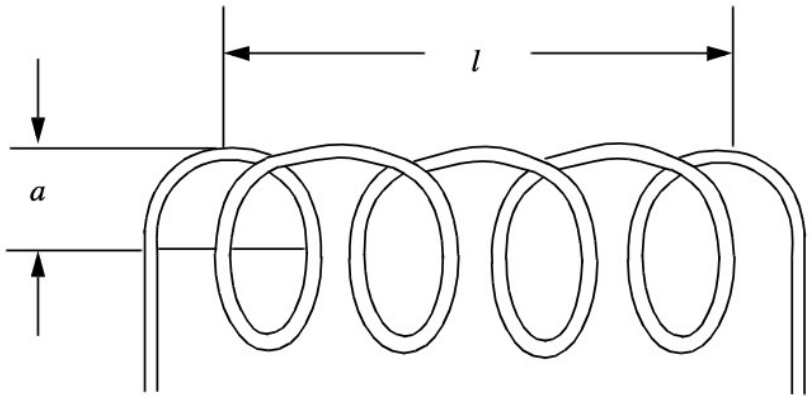

Рисунок 4.2 – Котушка індуктивності

Індуктивність котушки індуктивності, зображеної на рисунку 4.2, можно обрахувати за формулою 4.2

$$L = \frac{(aN)^2}{22.9l + 25.4a} \tag{4.2}$$

Де, a – радіус котушкиl;

l – довжина котушки;

N – кількість витків у котушці.

Антенну котушку для RFID-мітки можна налаштувати багатьма різними способами, залежно від мети застосування та розмірних обмежень. Типова індуктивність L для котушки мітки становить кілька мГн для пристроїв 125 кГц, саме така частота котушки використовується в цій роботі. На рисунку 4.3 показані різні конфігурації котушок для RFID-міток. Котушка зазвичай виготовляється з тонкого дроту.

Індуктивність і кількість витків котушки можна розрахувати за формулами, наведеними вишче.

Типова кількість витків котушки знаходиться в діапазоні 100 витків для 125 кГц і 3~5 витків для пристроїв 13,56 МГц.

Для більшого діапазону зчитування котушка антени повинна бути належним чином налаштована на цікаву частоту (тобто 125 кГц). Падіння напруги на котушці максимізується за рахунок формування паралельного резонансного контуру.

Для більшої ефективності – налаштування рекомендується виконувати за допомогою резонансного конденсатора, який підключений паралельно до котушки, як зображено на рисунку 4.3 [7].

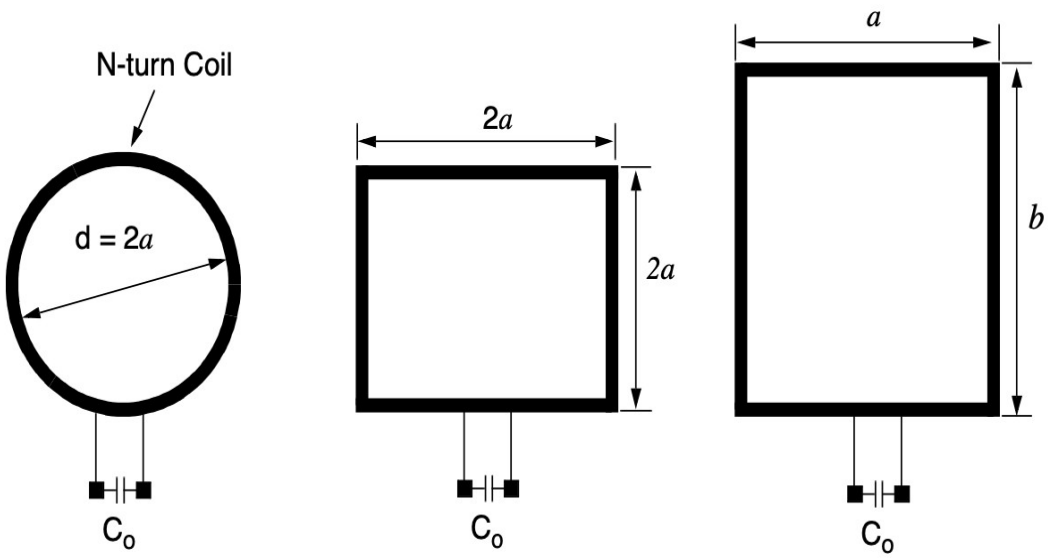

Рисунок 4.3 – Різні варіації котушок для RFID-міток

Для завантаження програми в енергонезалежну пам’ять програми мікроконтролера ATtiny85 треба послідовно з’єднати його виходи з виходами обраного USBASP програматора (рисунок 3.3). Схема розташування контактів мікроконтролера ATtiny85 наведено на рисунку 1.2, а схему контактів USBASP програматора наведено на рисунку 1.5.

Програматор має бути встановлено в режим програмування сторонніх пристроїв, для цього роз’єм JP1 має бути розімкнено. Роз’єм JP3 також встановлюємо у розімкнене положення для роботи програматора у режимі високої частоти. Перемичку на порту JP2 встановлюємо у положення живлення програматору від 5В. Програматор USBASP дозволить інженеру отримати простий, компактний і надійний програматор всіх мікроконтролерів сімейства AVR з режимом послідовного програмування ISP (In System Programming). Це зручний і мініатюрний програматор, що підключається до USB-порту персонального комп'ютера, що дуже актуально, через те що COM-порт існує далеко не на всіх сучасних комп'ютерах, і тим більше на ноутбуках.

Загрузка розробленої програми в пам’ять мікроконтролера буде здійснюватись за допомогою програми Arduino IDE.

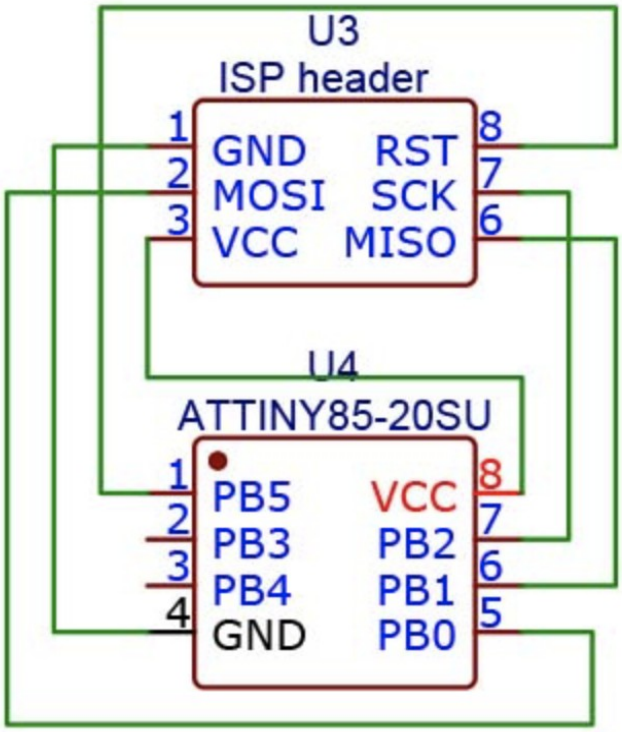

Рисунок 4.4 – Схема підключення USBASP програматора до мікроконтролера ATtiny85

4.2 Проектування та реалізація програмної частини проекту

Розробка додатку атестаційної роботи починається з системних функцій void setup() та void loop(). Усі комнади, директиви та налаштування мікроконтролера слід виконувати в середині функції setup(), вони будуть виконані один раз під час старту системи та ігноруватися під час подальшого виконання програми.

Після виконання коду прописаного в середині функції setup() починає виконуватись код функції loop(). Почав виконання з першої команди, мікроконтролер поступово дійде до останньої та почне виконання з початку. void loop() – це головна функція, точка входу в нашу програму, котра виконується тисячі раз в секунду.

Як зазначено в четвертому розділі, тип драйверу обраного дісплейного

модуля – контролер SSD1306. Для зручності в ході розробки програми на початку файлу було оголошено декілько глобальних (лістинг 4.1).

Лістинг 4.1 – Оголошення змінних для роботи з дісплейним модулем

```
#define OLED_ADDRESS            0x3C
#define OLED_COMMAND_MODE       0x00
#define OLED_ONE_COMMAND_MODE   0x80
#define OLED_DATA_MODE          0x40
#define OLED_ONE_DATA_MODE      0xC0
```

У лістингу 4.1 було об’явлено п’ять основних змінних, котрі будуть використовуватись для роботи з дісплейним модулем. За допомогою директиви #define була визначена послідовність символів, яка буде заміщатися значенням адреси контролера SSD1306 у коді програми. Розглянемо об’явлені змінні детальніше:

- OLED_ADRESS – адреса для спілкування по I2C протоколу;
- OLED_COMMAND_MODE – перевести контролер дісплейного модуля у режим считування команд;
- OLED_ONE_COMMAND_MODE – перевести контролер дісплейного модуля у режим читання однієї команди;
- OLED_DATA_MODE – перевести контролер дісплейного модуля у режим читання байт данних;
- OLED_ONE_DATA_MODE – перевести контролер дісплейного модуля у режим читання одного байта данних;

Завдяки переліченим вишче змінним мікроконтролер ATtiny85 може спілкуватися з дісплейним модулем. Проте перед безпосереднім використанням дісплея його треба налаштувати.

Налаштування дісплейного модуля здійснюється завдяки зарезервованим словам-командам. Серію команд для налаштування дісплейного модуля для даної атестаційної роботи було винесено в окрему змінну масив, наведену в лістингу 4.2.

Лістинг 4.2 – Серія команд для налаштування дісплейного модуля

```
const uint8_t oled_init_sequence[] PROGMEM = {
  0xAE,          // oled off
  0xD5,          // CLOCK_DIV_RATIO
  0x80,
  0xA8,          // Set multiplex
  0x3F,          // for 64 rows
  0xD3,
  0x00,         // Set display offset. 00 = no offset
  0x40 | 0x00,  // Set start line address, at 0.
  0x8D,          // Charge pump
  0x14,
  0x20,          // Memory mode
  0x00,          // Vertical OLED_ADDRESSing
  0xA1,          // Flip horizontally
  0xC8,          // Flip vertically
  0x81,          // Set contrast
  0xCF,          // brighter
  0xDB,          // Set vcom detect
  0x40,          // brighter
  0xDA,
  0x12,
  0xAF,          // Display on
};
```

У лістингу 4.2 наведені команди для первинного налаштування дісплейного модуля. Можно виділити команди включення, виключення, установка кількості рядків матриці, установка початкової точки, завдання контрасту і яскравості, установка режиму адресації матриці, завдання зсуву та інщі.

Змінна oled_init _sequence, об’явлена у лістингу 4.2, лише зберігається у енергонезалежній пам’яті мікроконтролера. Для безпосередньої ініціалізації дисплейного модуля перерахованими вище командами було написано функцію oled_init() (лістинг 4.3).

Функція oled_init(), не приймає жодних параметрів, та вертає тип даних void. Спочатку переводить дісплейний модуль у режим считування команд, а потім у циклі поступово зчитує команди, що зберігаються у змінній oled_init_sequence, та посиляє їх на контролер дісплейного модуля, тим самим проводить його налаштування.

Виклик функції oled_init() здійснюється лише один раз за період життєвого циклу програми в середині функції setup(), котра описується в розділі 3.2.1.

Лістинг 4.3 – Код функції oled_init() для налаштування дісплейного модуля

```
void oled_init() {
  Wire.beginTransmission(OLED_ADDRESS);
  Wire.write(OLED_COMMAND_MODE);

  for (uint8_t i = 0; i < sizeof(oled_init_sequence); i++)
            Wire.write(pgm_read_byte(&oled_init_sequence[i]));

  Wire.endTransmission();
}
```

Для роботи з вже налаштованим дісплейним модулем були розроблені функції. Нижче будуть розглянуті основні з них, програмний код та приклади використання будуть наведені в прикладах нижче.

Функція void oled_write_byte(byte b), котра приймає байт данних як параметр та посилає його на дісплейний модуль (лістинг 4.4). Функція вертає тип даних void.

Лістинг 4.4 – Код функції oled_write_byte() для відправки данних на дісплейний модуль

```
void oled_write_byte(byte b) {
  Wire.beginTransmission(OLED_ADDRESS);
  Wire.write(OLED_DATA_MODE);
  Wire.write(b);
  Wire.endTransmission();
}
```

Як показано у лістингу 4.4, функція oled_write_byte() спочатку переводить контролер дісплейного модуля у режим читання данних і після цього посилає байт.

Дана функція широко використовується в атестаційній роботі для безпосереднього відображення графічної інформації на дісплеї. Для відображення мінімальною одиницею інформації було обрано байт, а не скажімо біт, через зручність в використанні та пакуванні даних.

Функція void oled_set_pos(uint8_t x, uint8_t y), котра встановлює активний

курсор дісплейного модуля за координатами x та y відповідно (лістинг 4.5).

Лістинг 4.5 – код функції oled_set_pos() для завдання позиції курсора на екрані дісплейного модуля

```
void oled_set_pos(uint8_t x, uint8_t y) {
  Wire.beginTransmission(OLED_ADDRESS);
  Wire.write(OLED_COMMAND_MODE);

  Wire.write(0xb0 | y);
  Wire.write(x & 0xf);
  Wire.write(0x10 | (x >> 4));

  Wire.endTransmission();
}
```

Як вже було зазначено у попередніх розділах, для даної кваліфікаційної роботи було обрано дісплейний модулю із щільністю екрану 128 на 64 пікселя. Тобто, для нормального відображення графічної інформації дісплейним модулем, у якості параметрів uint8_t x та uint8_t y слід передавати значення, що не перевищують показники щільності екрану.

Функція oled_clear() – призначена для очищення дісплейного модул від графічної інформації (лістинг 4.6).

Лістинг 4.6 – код функції oled_clear() для очищення екрану дісплейного модуля

```
void oled_clear() {
  for (int i = 0; i < 128; i++) {
    Wire.beginTransmission(OLED_ADDRESS);
    Wire.write(OLED_DATA_MODE);
    for (int i = 0; i < 64; i++) Wire.write(0);
    Wire.endTransmission();
  }
}
```

Як показано у лістингу 4.6, функція oled_clear() для очищення екрану в циклі поступово проходить по всім пікселям екрану та задає їм значення 0, тобто переводить їх у вимкнений стан пікселю.

Для очищення екрану в методі використовується ітеративний цикл for. Цикли з використанням операторів for є однією з найважливіших конструкцій

мови С, що лежить в основі даної програми. Для повної очистки дісплея цикл пройде 128 ітерацій, що дорівнює кількості пікселів надісплейному модулі по горизонталі

Варто зазначити, що функцію oled_clear() потрібно викликати на етапі налаштування дісплейного модулю, в середині функції setup(), після виконання oled_init(). Це потрібно робити задля того, щоб прибрати з екрану усі «шуми», залишкову інформацію та приготувати його для роботи.

5. ПРОЦЕС ЕМУЛЯЦІЇ RFID-КЛЮЧА

5.1 Формат передачі даних між RFID-міткою та зчитувачем

Процес спілкування RFID-мітки зі зчитувачем досить простий проте в той самий час дуже елегантно побудований. Коли мітка хоче надіслати логічний нуль до зчитувача, він «навантажує» свою лінію живлення, щоб запитати більше енергії від зчитувача. Це призведе до невеликого падіння напруги на стороні зчитувача RFID. Цей рівень напруги є логічним нулем (лістинг зображено на рисунку 5.1). Одночасно, поки зчитувач передає сигнал 125 кГц, він зчитує напругу переданого сигналу через фільтри D1, C3 і R5, C1. Коли тег знижує напругу, як було сказано раніше, зчитувач фіксує це падіння напруги як логічний нуль. Коли тег не потребує додаткового живлення, він не викликає падіння напруги, це логіка одиниця (лістинг наведено на рисунку 5.1). Довжина «Одиниць» або «Нулів» залежить від швидкості послідовної передачі даних.

Варто наголосити, що швидкість передачі даних не є пропорційною до частоти котушки індуктивності, тобто для котушки з частотою в 125 кГц ми не будемо мати швидкість передачі даних в 125000 біт за секунду. Передача даних від тега до зчитувача варіюється від 500 біт в секунду до 8000 біт в секунду.

Хоча всі дані передаються хосту шляхом амплітудної модуляції (модуляція зворотного розсіювання), фактична модуляція одиниць та нулів виконується за допомогою трьох додаткових методів модуляції.

Пряма модуляція. У прямій модуляції амплітудна модуляція підходу зворотного розсіювання є єдиною застосовуваною модуляцією. Верхня для одиниці, а низька для нуля. Пряма модуляція може забезпечити високу швидкість передачі даних, але низький рівень захисту від побочних виникаючих шумів;

Модуляція частотним зсувом (Frequency Shift Keying). Ця форма

модуляції використовує дві різні частоти для передачі даних. Найпоширенішим режимом FSK є 8/10. Іншими словами, логічний нуль передається як амплітудно-модульований тактовий цикл з періодом, що відповідає частоті, поділеній на 8, а логічна одиниця передається як амплітудно-модульований період тактового сигналу, що відповідає несучій частоті поділеній на 10. Таким чином, амплітудна модуляція перемикається з однієї частоти на іншу, що відповідає логічним нулям або одиницям в бітовому потоці, і зчитувач повинен лише підраховувати цикли між виявленими піками тактових частот, щоб декодувати дані. FSK забезпечує просту конструкцію зчитувача, забезпечує дуже сильну стійкість до шумів, але страждає від нижчої швидкості передачі даних, ніж деякі інші форми модуляції даних. Нижче, на рисунку 5.1, зображено приклад модуляції даних за допомогою методики FSK;

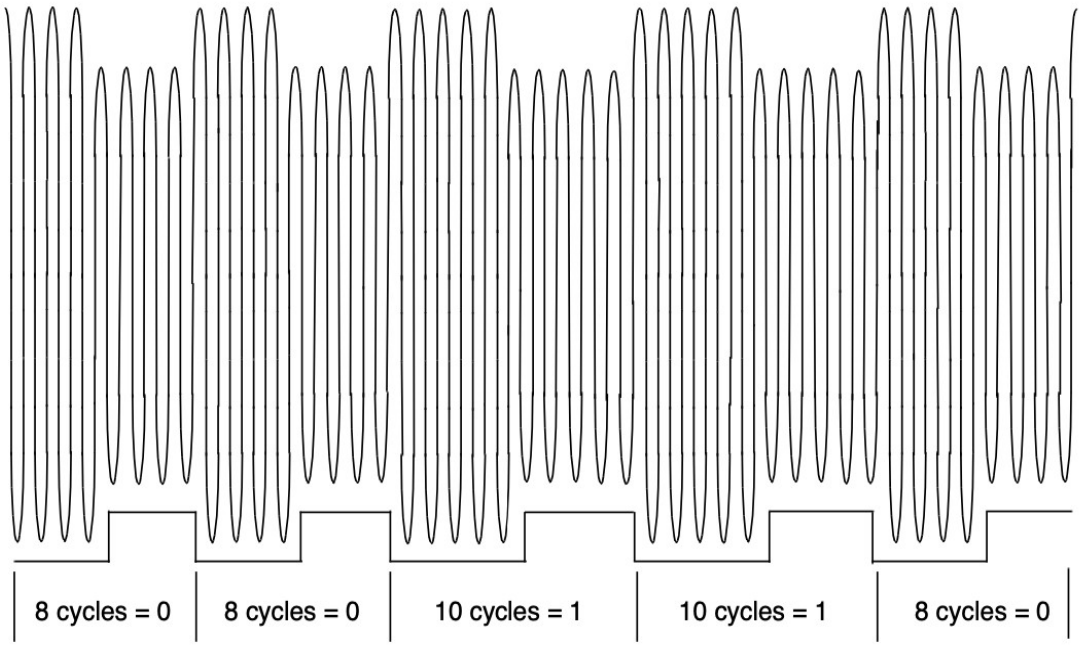

Рисунок 5.1 – Модуляція сигналу при передачі інформації до считувача

Модуляція фазовим зсувом (Phase Shift Keying). Цей метод модуляції даних подібний до FSK, за винятком того, що використовується лише одна частота, а зсув між логічною одиницею і нулем здійснюється шляхом зсуву фази

тактового сигналу зворотного розсіювання на 180 градусів. В свою чергу модуляція фазовим звуком теж поділяється на два підходи: зміна фази на будь-якому нулі, або зміна фази при будь-якій зміні даних (з одиниці на нуль та навпаки). Нижче, на рисунку 5.2 зображено приклад модуляції сигналу фазовим зсувом.

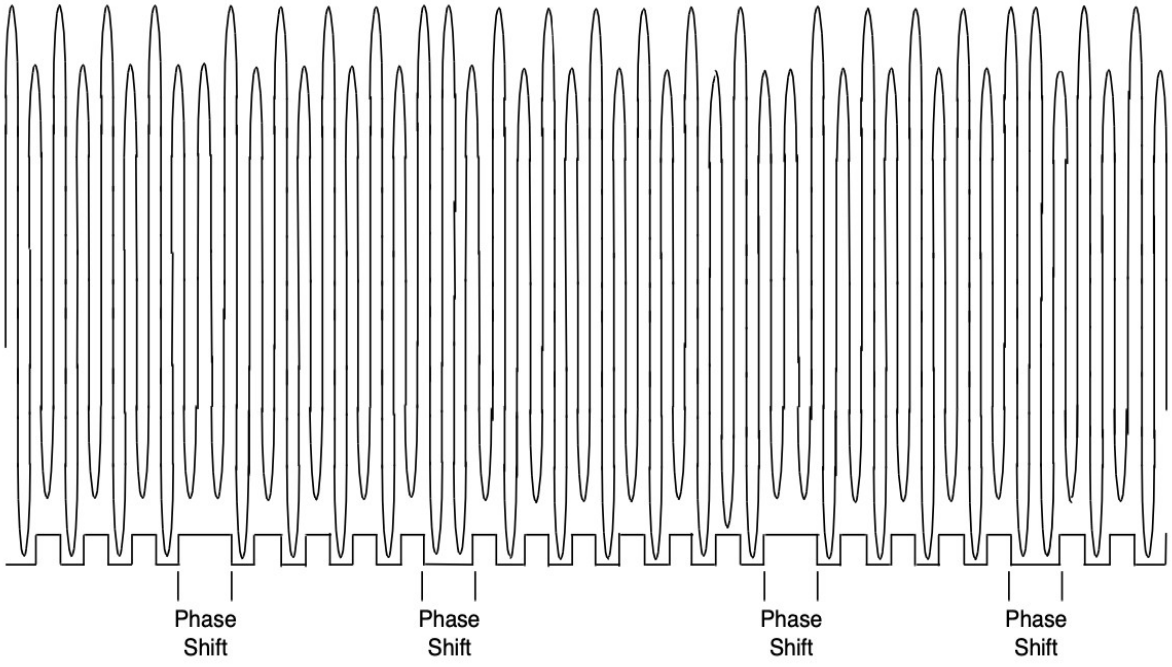

Рисунок 5.2 – Модуляція сигналу шляхом фазового зсуву

В пам’яті RFID-мітки зберігається не тільки зашифрована інформація самого “ключа”, мітка містить складну структуру даних для ініціювання обміну даними зі зчитувачем згідно із протоколом ISO 14223. Нижче, у таблиці 5.1, схематично наведено дані, котрі містяться в пам’яті мітки.

При розгляді даних таблиці будемо оперувати такими поняттями як стовпчики та рядки, не зважаючи на те, що в пам’яті програми усі дані будуть зберігатися в лінійному масиві. Також усім бітам даних в таблиці дано умовні позначення за допомогою букв латинської абетки, це значно спрощує опис та взаємодію із даними.

Таблиця 5.1 – Схематична пам’ять RFID-мітки

1	1	1	1	1	1	1	1	1
				D00	D01	D02	D03	P0
				D04	D05	D06	D07	P1
				D08	D09	D10	D11	P2
				D12	D13	D14	D15	P3
				D16	D17	D18	D19	P4
				D20	D21	D22	D23	P5
				D24	D25	D26	D27	P6
				D28	D29	D30	D31	P7
				D32	D33	D34	D35	P8
				D36	D37	D38	D39	P9
				PC0	PC1	PC2	PC3	S0

Розглянемо докладніше з чого складається 64 біти пам’яті RFID-мітки частотою на 125 кГц:

– перші 9 біт є початковими бітами зв'язку (завжди логічна одиниця);
– наступні 4 біти — це молодші значущі біти ідентифікатора клієнта (D00, D01, D02, D03);
– наступний 1 біт (P0) - це біт з контрольною сумою попередніх чотирьох біт;
– наступні 4 біти є старшими бітами ідентифікатора клієнта (D04, D05, D06, D07);
– наступний 1 біт (P1) - це біт з контрольною сумою попередніх чотирьох біт;
– наступні 4 біти є першою частиною серійного номера 32-бітового ключа (D08,...,D11);
– наступні біти починаючи з D12 та закінчуючи D39 є продовженням серійного номера 32-бітового ключа;

– біт PC0 — це біт парності бітів D00, D04, D08, D12, D16, D20, D24, D28, D32 і D36 (біти в одному стовпці);
– біти PC1, PC2, PC3 представляють біти парності наступних 3 стовпців;
– біт S0 – сигналізує про кінець передачі даних.

5.2 Програмна емуляція

Суть проекту полягає в тому, щоб мати можливість зберігати багато зліпків цифрових ключів в пам’яті одного компактного пристро, та мати можливість обирати потрібний ключ в окремих випадках. Процес емуляції починається з того, що потрібно дізнатись код оригінального ключа, після чого в коді програми ми можемо створити змінну, котра буде зберігати цифровий зліпок ключа. В лістингу 5.1 наведено створення змінних з кодами ключів умовного юзера для дома, роботи, дома батьків, тощо.

Лістинг 5.1 – Оголошення змінних з цифровими зліпками ключів

```
#define DOM                 0xFFFFFFFFFF
#define RABOTA              0xFFFFFFFFFF
#define RODITELY            0xFFFFFFFFFF
#define METAKOM_CYFRAL      0xFFFFFFFFFF
#define METAKOM_1           0x365A1140BE
#define CYFRAL_1            0x01FFFFFFF
#define VIZIT_1             0x565A1140BE
#define VIZIT_2             0x365A398149
#define ELTIS               0x0
#define LIFT                0x0B57814601
```

Для взаємодії з користувачем приладом використовується дисплей діагоналлю 0,96" виконаний за технологією OLED. Дісплей виконаний з 128x64 окремих OLED пікселів та може виводити як примітивну графічну так і текстову інформацію. У якості інтерфейсу прилад буде виводити на дісплейний модуль список з усіх ключів, що зберігаються в пам’яті мікроконтролера. Для відображення текстового списку треба створити масив із назвами перелічених вище ключів (лістинг 5.1) для графічного відображення на дісплеї.

Лістинг 5.2 – Масив з назвами ключів для відображення на дісплеї

```
uint64_t universalNames[] =
{
      DOM,
      RABOTA,
      RODITELY,
      METAKOM_CYFRAL,
      METAKOM_1,
      CYFRAL_1,
      VIZIT_1,
      VIZIT_2,
      ELTIS,
      LIFT
};
```

Для безпосередньої емуляції RFID-мітки – в коді програми було створено змінну для формування даних обраного ключа перед відправкою на зчитувач. Змінна буде задана як масив розміром у 128 байт та назвою RFIDdata. Ця змінна буде формувати не тільки дані обраного ключа, а й усю необхідну інформацію для успішної комунікації RFID-мітки з частотою у 125 kHz із зчитуючим приладом згідно із специфікацією стандарту RFID, ISO 14223. Формат та послідовність даних, що будуть формуватися в змінній RFIDdata – наведено в таблиці 5.1.

Лістинг 5.3 – Створення змінних для процесу емуляції

```
byte facility[2] = { 0x02, 0x0C };
byte cardID[8] = { 0x00, 0x00, 0x00, 0x00,
                   0x00, 0x00, 0x00, 0x00 };

int columns[4] = { 0, 0, 0, 0};

int bittime = 256;

byte RFIDdata[128];

byte datapointer = 0;

int clock = 0;
byte state;
```

Розглянемо інших змінних наведених у лістингу 5.3, варто розібрати:

– змінна datapointer – службова змінна для ітерації масивом та зберігання в пам’яті положення елементів;
– змінна bittime – у мікросекундах задає час паузи між сусідніми бітами інформації сигналу при надсиланні на RFID считувач. Для здійснення

паузи використовується штатна функція delayMicroseconds;

- змінна facility – використовується для зберігання молодших та старших значущіх бітів ідентифікатора клієнта. Біти D00 – D07, проілюстровані у таблиці 5.1;
- змінна cardID – масив на вісім значень використовується для зберігання безпосереднього коду ключа RFID-картки. Біти D08 – D39, проілюстровані у таблиці 5.1;
- змінна columns – використовується для проміжного зберігання контрольних сум біт в одному стовпці за виключенням перших дев’яти біт, котрі слугують своєрідним “заголовком”. Біти P00 – P03, проілюстровані у таблиці 5.1, до прикладу біт P00 зберігає контрольну суму першого стовпчика із бітами D00, D04, D08, D12, D16, D20, D24, D28, D32 і D36, за аналогією P01, P02 та P03 – зберігають контрольні суми у своїх стовпцях.

Тепер користуючись змінними з прикладу 5.3 та структурою пам’яті RFID-картки, наведеної у таблиці 5.1, необхідної для успішної комунікації між клюем та ситуваем можно розглянути усі розроблені функції для емуляції роботи RFID-картки.

Розглянемо функцію WriteHeader, наведену у лістингу 5.4. Як видно з її сигнатури, вона не приймає ніяких параметрів та вертає тип даних void, тобто не вертає даних в загалі. Головна та єдина мета функції – сформувати так званий “заголовок” даних. Як вже було показано у таблиці 5.1, для успішної комунікації між ключем та зчитувачем – перщі 9 біт сигналу мають бути встановлені в логічну одиницю.

Лістинг 5.4 – функція WriteHeader

```
void WriteHeader(void)
{
  RFIDdata [datapointer++]=1;
  RFIDdata[datapointer++]=1;
  RFIDdata[datapointer++]=1;
  RFIDdata[datapointer++]=1;
  RFIDdata[datapointer++]=1;
```

```
        RFIDdata[datapointer++]=1;
        RFIDdata[datapointer++]=1;
        RFIDdata[datapointer++]=1;
        RFIDdata[datapointer++]=1;
      }
```

Далі розглянемо допоміжний метод WriteData, цей метод займається формуванням даних у форматі прийнятному для RFID зчитувача. Метод формує молодші та старші значуші біти ідентифікатора клієнта (біти D00 – D07, проілюстровані у таблиці 5.1), а також код ключа RFID-картки (біти D08 – D39, проілюстровані у таблиці 5.1). Також метод бере на себе відповідальність за прораховуваня та запис контрольних сум для рядків бітів (біти P0 – P9, таблиця 5.1). Код метода WriteData наведено нижче у лістингу 5.5.

Лістинг 5.5 – Метод WriteData

```
void WriteData(byte input)
{
  byte data;
  byte rowsum=0;
  for (int i=4; i>0; i--)
  {
    if ((input& 1<<i-1) ==0)
          {
            data=0;
 }
    else
 {
    data=1;
    rowsum++;
    colsum[i-1]++;
}

 RFIDdata[datapointer++]= data;
 #ifdef SERIALDEBUG
  Serial.print((int) data);
#endif

  }
 // write the row checksum out
 if ((rowsum%2)==0)
 {
  RFIDdata[datapointer++]=0;
  #ifdef SERIALDEBUG
   Serial.print((int)0);
 #endif

 }
 else
 {
   RFIDdata[datapointer++]=1;
   #ifdef SERIALDEBUG
 Serial.print((int)1);
```

```
#endif
 }

   #ifdef SERIALDEBUG
    Serial.println();
  #endif
}
```

Після формування безпосередньо коду ключа, залишається лише сформувати біти парності для стовпчиків (P0, P1, P2, P3 – наведені в таблиці 5.1). Для цього був створений метод WriteChecksum (лістинг 5.6). Цей метод поступово інтегрується по масиву даних RFIDdata та розраховує біти парності для стовпчиків бітів D00 – D39.

Після вдалого розрахунку біти парності додаються в кінець масиву даних. Варто відмітити, що метод WriteChecksum працює з бітами вже сформованими на попередніх кроках програми, тобто дуже важливо зберігати послідовність виклику функцій та рухатись від бітів з номером D00 до D39 відповідно. Для цього має зберігатись послідовність виклику функцій, про що буде докладніше сказано далі.

Функція за допомогою циклу for робить чотири ітерації, що дорівнює кількості стовпчиків в таблиці 5.1, та за допомогою оператору залишку від ділення формує біти парності. Вміст тимчасової змінної columns за допомогою оператору залишку від ділення перевіряється на парність / непарність, шляхом знаходження залишку при поділі на два. Якщо залишок після поділу на два дорівнює нулю – то змінна парна, у протилежному віпадку – непарна. В залежності від результату попередньої операції в якості контрольної суми до масиву даних RFIDdata записується логічний “нуль” або логічна “одиниця” відповідно.

Якщо подивитись на метод WriteChecksum, наведений у лістингу 5.6, то можна відмітити, що він займається не лише формуванням бітів парності для чотирьох стовпчиків даних. В самому кінці метода, після ітеративного циклу for, в сформований масив даних для відправки на считува добавляється останній бітдорівнює логічному нулю.

Цей біт умовно називається “стоп” бітом та сигналізує RFID считовачу про кінець передачі даних. Разом за першими дев’яти бітами, котрі встановлені в логічну одиницю, та слугують умовним “заголовком”, останній біт виступають у ролі обмеження повідомлення закодованого у RFID-мітці для відправленя на сторону зчитувача (таблиця 5.1).

Лістинг 5.6 – Метод WriteChecksum

```
void WriteChecksum(void)
{
 byte data;
 byte rowsum=0;
 for (int i=4; i>0; i--)
  {

  if ((colsum[i-1]%2) ==0)
  {
    RFIDdata[datapointer++]=0;
   #ifdef SERIALDEBUG
    Serial.print((int)0);
   #endif
  }

  else
  {
    RFIDdata[datapointer++]=1;
     #ifdef SERIALDEBUG
     Serial.print((int) 1);
     #endif
   }
 }

 // write the stop bit
 RFIDdata[datapointer++]=0;

 #ifdef SERIALDEBUG
 Serial.print((int)0);
  #endif
}
```

Після прикладів наведених вище у цьому розділі, можна розглянути метод BuildCard. Цей метод є фінальним у ланцюжку побудови та формування даних на відправку від цифрового ключа, що емулює RFID-картку, на сторону зчитучого пристрою.

Як показано у лістингу 5.7, метод BuildCard, використовує функції, котрі були вже описані раніше (лістинг 5.4, лістинг 5.5, лістинг 5.6). Також, не зважаючи на те, що функція не приймає жодних параметрів, в середині він все ж таки звертається до деяких змінних, котрі були оголошени глобально.

Розглянемо докладніше цей метод у порядку його виконання:

- присвоєння значення нуль змінній datapointer. Ця службова змінна використовується у програмі для ітерації масивом та зберігання в пам’яті положення елементів. Присвоєння їй знаення нуля свідчить про початок нового циклу запису;
- метод WriteHeader використовується для формування та запису перших дев’яти біт заголовку. Біти заголовка встановлюються в логічну одиницю та разом із останнім “стоп” бітом виступають у ролі обмеження повідомлення закодованого у RFID-мітці для відправленя на сторону зчитувача (таблиця 5.1);
- метод WriteData в загальній кількості виконується десять разів. Проте його можна логічно розділити на дві групи. Перщі два виконання методу використовується для запису молодших та старших значущіх бітів ідентифікатора клієнта (біти D00 – D07). Наступні ж вісім виконань послуговують для формування та запису бітів серійного номера 32-бітового ключа (біти D08 – D39). Приклад результуючої структури даних наведено у таблиці 5.1;
- метод WriteChecksum використовується для запису контрольних сум біт в одному стовпці за виключенням перших дев’яти біт, котрі слугують своєрідним “заголовком”. Біти P00 – P03, проілюстровані у таблиці 5.1, до прикладу біт P00 зберігає контрольну суму першого стовпчика із бітами D00, D04, D08, D12, D16, D20, D24, D28, D32 і D36, за аналогією P01, P02 та P03 – зберігають контрольні суми у своїх стовпцях.

Лістинг 5.7 – Метод BuildCard

```
void BuildCard(void)
{
// load up the RFID array with card data
// intitalise the write pointer
datapointer=0;

WriteHeader();
```

```
// Write facility
WriteData(facility[0]);
WriteData(facility[1]);

// Write cardID
WriteData(cardID[0]);
WriteData(cardID[1]);
WriteData(cardID[2]);
WriteData(cardID[3]);
WriteData(cardID[4]);
WriteData(cardID[5]);
WriteData(cardID[6]);
WriteData(cardID[7]);

WriteChecksum();
}
```

Після усіх підготовчих робіт з формування диних в масиві змінної RFIDdata, можно нарешті перейти безпосередньо до процесу відправки даних із сторони побудованого ключа на сторону RFID считувача. Для цього в програмі використовується метод EmulateCard, наведений у лістингу 5.8. Розглянемо цей метод докладніше.

З самого початку функція функція викликає метод BuildCard, наведену у лістингу 5.7, для формування послідовності даних для відправки на сторону зчитуючого пристрою. Після того, як данні сформовані та записані в тимчасову змінну RFIDdata, починається безпосередньо процес спілкування пристрою із считувачем.

В середині нескінченного циклу while, знаходиться ще один ітеративній цикл for. В середині цього циклу поступово виконується команда TransmitManchester, котра буде розглянута нижче у лістингу 5.9. Команад бере біти інвормації, котрі були попередньо сформовані та збережені у тимчасовій змінній, та посилає їх на сторону считувача. Із кожною наступною ітераціє цикла, береться наступний за номером біт інформації.

Лістинг 5.8 – Метод EmulateCard

```
void EmulateCard(void)
{
#ifdef SERIALDEBUG
Serial.println("Emulate Card Entered");
#endif
```

```
BuildCard();

#ifdef SERIALDEBUG
Serial.println();
for(int i = 0; i < 64; i++)
{
      if (RFIDdata[i]==1) Serial.print("1");
      else if (RFIDdata[i]==0) Serial.print("0");
      else Serial.print((int)RFIDdata[i]);
}
Serial.println();
#endif


while (1)
{
for(int i = 0; i < 64; i++)
{
TransmitManchester(0, RFIDdata[i]);
      delayMicroseconds(bittime);
      TransmitManchester(1, RFIDdata[i]);
      delayMicroseconds(bittime);
}
}
}
```

Цикли з використанням операторів for та while є однією з найважливіших конструкцій мови С, що лежить в основі даної програми. Вони трапляються абсолютно в кожному скечті.

Оператор WHILE використовується в С для організації повтору одних і тих команд довільну кількість разів. У порівнянні з FOR цикл WHILE виглядає простіше, він зазвичай використовується там, де не потрібен підрахунок числа ітерацій у змінній, а просто потрібно повторювати код, доки щось не зміниться, не настане якась подія. Як умови може використовуватися будь-яка конструкція мови, що повертає логічне значення. Умовами може бути операції порівняння, функції, константи, змінні. Як і за будь-яких інших логічних операціях будь-яке значення, крім нуля сприйматиметься як істина або логічна одиниця, нуль – брехня логічний нуль.

Розглянемо докладніше метод TransmitManchester, наведений нижче у лістингу 5.9. Головне та єдине його призначення – передача даних на котушку інуктивності.

Функція digitalWrite - одна з найпопулярніших функцій в апаратних проектах. Вона зустрічається практично в будь-якому проекті, де потрібно

поблимати світлодіодами або подати якісь сигнали на підключені пристрої.

Функція дозволяє керувати підключеним обладнанням шляхом подачі або зняття робочої напруги (у більшості випадків – 5В) на піни. По суті, digitalWrite перетворює контролер на велику розумну розетку, що включає або вимикає напругу на її портах залежно від наших завдань. Подаючи напругу на підключені пристрої, ми змушуємо щось робити: наприклад, включати світло, видавати звуки, включати двигун. Крім того, за допомогою digitalWrite можна подавати імпульси та передавати інформацію закодованими сигналами та повідомленнями.

Контролер може сформувати лише цифровий сигнал у двійковому коді (логічна одиниця – висока напруга, логічний нуль – низька напруга). На відміну від аналогового сигналу, ми не можемо точно виставити напругу на виході, наприклад, 4,15 вольт. Можливі тільки два значення напруги: мінімальне і максимальне робоче (наприклад, для Arduino Uno це 0 В і 5 В відповідно). Природно, в реальному житті напруга не дорівнює 0 або 5 вольт, але при правильній схемі живлення плати Arduino, можна вважати всі можливі відхилення несуттєвими.

Функція приймає два параметри: номер піна контролера та значення нуля чи одиниці, проте краще використовувати зарезервовані константи HIGH та LOW відповідно. Функція не повертає жодного значення, а результат виконання залежить від вказаного первим параметром номера піна.

Якщо контакт був налаштований як OUTPUT за допомогою методу pinMode(), його напруга буде встановлено на відповідне значення: HIGH, та LOW відповідно.

Якщо контакт налаштовано як INPUT, метод увімкне (HIGH) або вимкне (LOW) внутрішнє підтягування на вхідному контакті. Рекомендується встановити для pinMode() значення INPUT_PULLUP, щоб увімкнути внутрішній підтягуючий резистор.

Якщо для pinMode() не було встановлено значення значення OUTPUT і не

було підключино світлодіод до контакту, під час виклику digitalWrite(HIGH) світлодіод може бути тьмяним. Без явного налаштування pinMode(), digitalWrite() увімкне внутрішній підтягуючий резистор, який діє як великий струмообмежувальний резистор.

В даному проекті у якості першого параметру функції було передано константу COIL, котра встановлена в значення 3. Значення змінних було встановлено на основі апаратної схеми пристрою (рисунок 4.1). Як ми бачимо на схемі, котушку індуктивності під'єднано до третього пінку мікроконтролера Attiny85 із номером PB4.

Лістинг 5.9 – Метод TransmitManchester

```
void TransmitManchester(int cycle, int data)
{

      if(cycle ^ data == 1)
      {
            digitalWrite(COIL, HIGH);
      }
      else
      {
            digitalWrite(COIL, LOW);
      }
}
```

Як видно з прикладу 5.9, в середині методу TransmitManchester використовується умовний оператор if-else. Умовні конструкції - один із базових компонентів багатьох мов програмування, які спрямовують роботу програми по одному зі шляхів залежно від певних умов. Однією з таких конструкцій у мові програмування C є конструкція if. Конструкція if/else перевіряє істинність певної умови та залежно від результатів перевірки виконує певний код.

Після ключового слова if ставиться умова. Умова повинна представляти значення bool. Це може бути безпосередньо значення типу bool або результат умовного виразу або іншого виразу, який повертає значення типу bool. І якщо ця умова є істинною (дорівнює логічній одиниці), то спрацьовує код, який поміщений далі після умови всередині фігурних дужок.

У деяких джерелах сказано, що оператор вибору if else – самостійний

оператор. Але це не так, if else — це лише форма запису оператора вибору if. Оператор if else дозволяє визначити програмісту дію, коли умова істинна та альтернативна дія, коли умова хибна. Тоді як if дозволяв визначити дію за справжньої умови.

Для всіх форм if інструкції, condition яка може мати будь-яке значення, крім структури, обчислюється, включаючи всі побічні ефекти. Управління передається з if оператора в наступний оператор у програмі, якщо не виконується if-branch або не else-branch. Оператор else пов'язаний з найближчим попереднім if оператором у тій же області, яка не має відповідної else інструкції.

За допомогою умовного оператора if, метод TransmitManchester, робіть вибір – буде посилатись на другий пін мікроконтролеру сигнал з логічним нулем або логічно одиницею відповідно до того чи умова блока if була істиною (логічною одиницею), чи логічним нулем.

У якості умови в операторі if використовується оператор побітового виключаючого АБО (булевий оператор XOR) між двома вхідними параметрами функції.

В програмуванні та математиці є певні операції для маніпуляцій з бітами. Ці операції називаються логічними або булевими операціями, названі на честь одного з математиків - Джорджа Буля , який сприяв розвитку цієї галузі науки.

Всі ці операції можуть бути застосовані до будь-якого біта, незалежно від того, яке він має значення - 0 (нуль) або 1 (одиницю). Нижче наведено основні логічні операції та приклади їх використання.

В курсі математичного аналізу вивчаються функції, визначені на числовій прямій або на відрізку числової прямої або площині і т.п. Так чи інакше область визначення – безперервна множина. У курсі дискретної математики вивчаються функції, область визначення яких – дискретна безліч. Найпростішою але нетривіальною такою множиною є безліч, що складається з двох елементів. Так і формується поняття булевої функції. Булевою функцією від n аргументів

називається функція f з n-ого ступеня множини {0, 1} до множини {0, 1}.

Інакше висловлюючись, булева функція – це функція, аргументи і значення якої належить множині { 0, 1 }. Безліч {0, 1} ми будемо надалі позначати через B.

Булеву функцію від n аргументів можна розглядати як n-місцеву операцію алгебри на множині B. При цьому алгебра <B; W>, де W - безліч всіляких булевих функцій, називається алгеброю логіки.

Кінцевість області визначення функції має важливу перевагу – такі функції можна задавати перерахуванням значень у різних значеннях аргументів. Для того, щоб встановити значення функції від n змінних, треба визначити значення для кожного з 2n наборів. Ці значення записують у таблицю у порядку відповідних двійкових чисел.

Працюючи з булевими функціями відбувається повне абстрагування від змістовного сенсу, який передбачається в алгебрі висловлювань. Проте між булевими функціями та формулами алгебри висловлювань можна встановити взаємно-однозначну відповідність, якщо:

- встановити взаємно-однозначну відповідність між булевими та позиційними змінними;
- встановити зв'язок між булевими функціями та логічними зв'язками;
- залишити пріоритет операцій без змін.

Практично всі булеві функції нижчих арностей (0, 1, 2 і 3) отримали імена, що історично склалися. Якщо значення функції не залежить від однієї зі змінних, то ця змінна, не граючи жодного значення, називається фіктивною.

Виключне «або», або складання за модулем 2, або XOR — булева функція, а також логічна і бітова операція, тоді й тільки тоді, коли один із аргументів є істинним, а інший — хибним. Для функції трьох (тернарне додавання за модулем 2) і більше змінних — результат виконання операції буде дійсним лише тоді, коли кількість аргументів, що дорівнює 1, що становлять поточний набір — непарна. Така операція природним чином виникає в кільці відрахувань за

модулем 2, звідки і походить назва операції.

Теоретично множин додаванню по модулю 2 відповідає операція симетричної різниці двох множин.

У булевій алгебрі додавання за модулем 2 - це функція двох, трьох та більше змінних, вони ж - операнди операції, вони ж - аргументи функції. Змінні можуть набувати значень з безлічі { 0, 1 }. Результат також належить множині {0, 1}.

Булева функція задається кінцевим набором значень, що дозволяє її у вигляді таблиці істинності. Обчислення результату проводиться за допомогою простому правилу, чи з таблиці істинності (таблиця 4.2). Замість значень 0, 1 може використовуватись будь-яка інша пара відповідних символів, наприклад false, true або F, T або «брехня», «істина», але при цьому необхідно визначати старшість, наприклад, true > false.

У мовах програмування C/C++, Java, C#, Ruby, PHP, JavaScript, Python тощо, бітова операція порозрядного доповнення позначається символом «^», у мовах Паскаль, Delphi, Ada, Visual Basic — зарезервованим словом xor, у мові асемблера - однойменною логічною командою. При цьому додавання по модулю 2 виконується для всіх бітів лівого та правого операнда попарно. Виконання операції, що виключає «або» для значень логічного типу true / false, проводиться в різних мовах програмування по-різному.

Наприклад, Delphi використовується вбудований оператор XOR. У мові C, починаючи зі стандарту C99, оператор «^» над операндами логічного типу повертає результат застосування логічної операції XOR.

У C++ оператор «^» для логічного типу bool повертає результат згідно з описаними мовою правилами, для інших типів проводиться його побітове застосування.

Використання побітового виключаючого «або» дозволяє поміняти місцями значення цілих змінних без використання додаткової пам'яті.

Таблиця 5.2 – Таблиця істинності оператора XOR

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Розглянемо докладніше параметр cycle функції TransmitManchester, та яка логіка стоїть за цим аргументом. Як вже розглядалось раніше, в попередніх розділах, для модуляції сигналу від RFID-катушки на считувач – використовується метод модуляції частотним зсувом.

Модуляція частотним зсувом (FSK) – це метод цифрової модуляції, при якому частота несучого сигналу змінюється відповідно до змін цифрового сигналу. FSK - це схема частотної модуляції.

Вихід FSK – модульованої хвилі має високу частоту для двійкового високого входу (логічної одиниці) і низьку частоту для двійкового низбкого входу (логічного нуля). Двійкові одиниці і нулі називаються частотами позначки і просторовими частотами.

Найпростішим FSK є двійковий або бінарний FSK (BFSK). BFSK використовує пару дискретних частот для передачі двійкової (нулі і одиниці) інформації. У цій схемі одиниця називається частотою позначки, а нуль — просторовою частотою.

На наведеному нижче зображенні (рисунок 5.1) показано діаграму модулованого сигналу FSK разом з його вхідним сигналом.

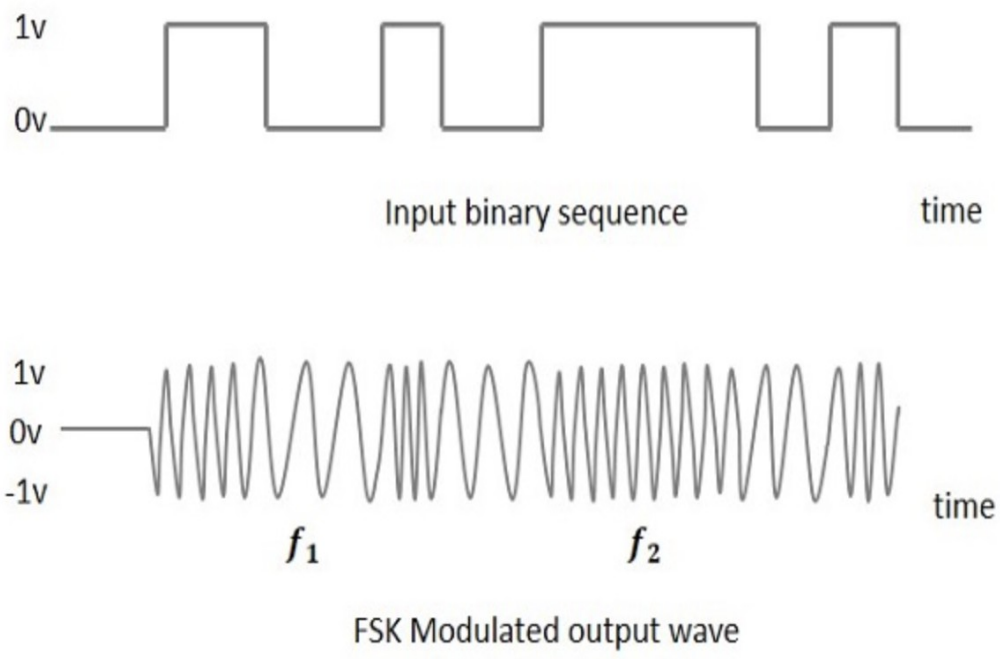

Рисунок 5.3 – Модуляція частотним зсувом та вхідний сигнал

Технологія модуляції частотним зсувом використовується для систем зв'язку, таких як телеметрія, радіозонди метеорологічних куль, ідентифікація абонента, механізми відкриття гаражних дверей і низькочастотна радіопередача в діапазонах VLF і ELF, та інших.

Принцип модуляції частотним зсувом може бути реалізований за допомогою повністю незалежних вільних осциляторів і перемикання між ними на початку кожного періоду нового символу. Загалом, незалежні осцилятори не будуть мати однакову фазу і, отже, однакову амплітуду в момент перемикання, що спричиняє раптові розриви в переданому сигналі, що значно шкодить загальній картині.

На практиці багато передавачів FSK використовують лише один генератор, і процес перемикання на іншу частоту на початку кожного періоду символу зберігає фазу. Усунення розривів у фазі (а, отже, усунення різких змін амплітуди) зменшує потужність бічної смуги, зменшуючи перешкоди сусіднім каналам.

Більшість ранніх модемів для телефонних ліній використовували аудіо-частотну модуляцію (AFSK) для надсилання та отримання даних зі швидкістю приблизно до 1200 біт на секунду. Модеми Bell 103 і Bell 202 використовували цю техніку. Навіть сьогодні північноамериканська ідентифікація абонента використовує 1200 бод AFSK у формі стандарту Bell 202. Деякі ранні мікрокомп'ютери використовували особливу форму модуляції AFSK, стандарт Канзас-Сіті, для зберігання даних на аудіокасетах. AFSK все ще широко використовується в радіоаматорі, оскільки дозволяє передавати дані через незмінене обладнання для голосового діапазону.

6 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

6.1 Процес налагодження програми та дебагінга

Налагодження – це етап розробки комп'ютерної програми, на якому виявляють, локалізують та усувають помилки в її роботі. Щоб зрозуміти, де виникла помилка, доводиться:

- дізнаватися про поточні значення змінних;
- з'ясовувати, яким шляхом виконувалася програма;
- існують дві взаємодоповнюючі технології налагодження;
- використання налагоджувачів – програм, які включають в себе інтерфейс для покрокового виконання програми: оператор за оператором, функція за функцією, зупинки на деяких рядках вихідного коду або при досягненні певної умови;
- виведення поточного стану програми за допомогою розміщених у критичних точках програми операторів виведення – на екран, принтер, гучномовець або файл. Виведення налагоджувальних відомостей файл називається логуванням.

Здібності програміста до налагодження – це, мабуть, найважливіший фактор у виявленні джерела проблеми, але складність налагодження сильно залежить від мови програмування та інструментів, зокрема, відладчиків. Відладчик є програмним інструментом, що дозволяє програмісту спостерігати за виконанням досліджуваної програми, зупиняти і перезапускати її, проганяти в уповільненому темпі, змінювати значення в пам'яті і навіть, у деяких випадках, повертати назад за часом.

Також корисними інструментами в руках програміста можуть бути:

- профілювальники. Вони дозволять визначити, скільки часу виконується та чи інша ділянка коду. Аналіз покриття дозволяє виявити

нездійснювані ділянки коду;

- API логери дозволяють відстежити взаємодію програми та Windows API за допомогою запису повідомлень Windows у лог;
- дизассемблери дозволяють подивитися асемблерний код виконуваного файлу;
- сніффери допоможуть відстежити мережевий трафік, що генерується програмою;
- сніфери апаратних інтерфейсів дозволяють побачити дані, якими обмінюються система та пристрій;
- лог системи.

Як відомо, середовище Arduino (AVR) не містить функції внутрішньосхемного налагодження, що створює великі незручності при пошуку складних помилок та супроводі проектів, проте ATtiny85, 8-бітний AVR мікроконтролер фірми Atmel, представник сімейства tiny – має USB-TTL конвертер, що дозволяє мікроконтролеру в текстовому режимі "консолі" передавати дані та спілкуватися з комп'ютером за допомогою послідовного інтерфейсу, Serial.

На комп'ютері створюється віртуальний COM порт, до якого можна підключитися за допомогою програм-терміналів порту, і приймати або надсилати текстові дані. Через цей порт завантажується прошивка, т.к. підтримка Serial є вбудованою в мікроконтроллер на "залізному" рівні, і USB-TTL перетворювач підключений саме до цих пінів мікроконтролера. У самому середовищі програмування Arduino IDE також є вбудована консоль - монітор порту, кнопка з іконкою лупи в правому верхньому кутку програми. Натиснувши на цю кнопку, можна відкрити сам монітор порту, в якому будуть налаштування:

а) кінець рядка – тут є декілька варіантів на вибір, проте краще поставити немає кінця рядка, тому що це дозволить уникнути незрозумілих помилок на перших етапах знайомства з платформою. Інщі варіанти наведені нижче:

а.1. немає кінця рядка – жодних додаткових символів наприкінці введених символів після натискання на кнопку відправлення/Enter;

а.2. NL – символ перенесення рядка наприкінці надісланих даних;

а.3. CR – символ повернення каретки наприкінці надісланих даних;

а.4. NL+CR – поєднання двох опцій;

б) швидкість – цей параметр можно достить глибоко налаштувати під свої потреби, т.к. спілкування по Serial може здійснюватися на різних швидкостях, що вимірюються в бод (baud), і якщо швидкості прийому та відправки не збігаються – дані будуть отримані некоректно. За замовчуванням швидкість дорівнює 9600, залишимо параметр за замовченням;

в) очистити висновок - тут все трівіально, очищає висновок.

Розглянемо докладныше одну один з найуживаніших механізмів для отладки коду, Serial. Serial – це об'єкт класу Stream, що дозволяє як просто приймати/надсилати дані через послідовний порт, так і успадковує з класу Stream купу корисних можливостей, розглянемо докладніше деякі з них:

- Serial.begin – запустити зв'язок по Serial на швидкості speed (baud rate, біт за секунду). Швидкість можна поставити будь-яку, але є кілька стандартних (300, 1200, 2400, 4800, та інщі);
- Serial.end – припинити зв'язок із Serial;
- Serial.available – повертає кількість байт, що зберігаються в буфері (обсяг буфера 64 байта) та доступні для читання;
- Serial.availableForWrite – повертає кількість байт, які можна записати до буфера послідовного порту, не блокуючи при цьому функцію запису;
- Serial.print(val) – відправляє порт значення val – число чи рядок. На відміну від write виводить саме символи, тобто Serial.print(88) – виведе 88. Також метод print/println має кілька налаштувань для різних даних, що робить його дуже зручним інструментом налагодження.

В програмному коду проекту ще на етапі розробки та проектування були

закладені можливості для його налаштування та дебагінгу. В більщій частині функцій, котрі описувались в попередніх розділах використовуються такі методи як Serial.println для виводи корисної для розробника інформації. Такий підхід допоміг скоротити час на розробку та тестування програмної частини та коректної роботи апаратного комплексу проекту. Проте дебагінг використовує не лише об’єкт класа Serial, в процесі налаштування програми було використано ще технологію препроцесорів.

Препроцесор - це спеціальна програма, що є частиною компілятора Сі. Вона призначена для попередньої обробки тексту програми. Препроцесор дозволяє включати в текст програми файли та вводити макровизначення. Робота препроцесора здійснюється за допомогою спеціальних директив (вказівок). Вони відзначаються знаком дієзу #. Після закінчення рядків, що позначають директиви в мові Сі, точку з комою можна не ставити.

Одна з цілей використання "умовної компіляції" - зробити програму мобільнішою. Змінюючи кілька ключових визначень на початку файлу, можна встановлювати різні значення та включати різні файли для різних систем.

Ці короткі приклади ілюструють дивовижну здатність мови Сі витончено та суворо керувати програмами. Нижче буде наведено приклади найпоширеніших умов компіляції:

- #include — вставляє текст із вказаного файлу;
- #define — задає макровизначення (макро) або символічну константу
- #undef — скасовує попереднє визначення;
- #if - здійснює умовну компіляцію при істинності константного виразу;
- #ifdef - Здійснює умовну компіляцію при визначеності символічної константи;
- #ifndef - здійснює умовну компіляцію при невизначеності символічної константи;
- #else - Гілка умовної компіляції при помилковості вираження;
- #elif - Гілка умовної компіляції, утворена злиттям else і if;

- #endif - кінець гілки умовної компіляції;
- #line — препроцесор змінює номер поточного рядка та ім'я файлу, що компілюється;
- #error - Видача діагностичного повідомлення;
- #pragma - Дія, що залежить від конкретної реалізації компілятора.

Використання умов компіляції допомагає оптимізувати програмний код та підвищити його ефективність. Директиви умовної компіляції препроцесора дозволяють компілювати або пропускати частину програми, залежно від виконання певної умови.

Варто додати пару слів про місце налагодження програмного коду в циклі розробки проекту. Загалом типовий цикл розробки, що за час життя програми багаторазово повторюється, виглядає приблизно так:

- програмування – внесення до програми нової функціональності, виправлення існуючих помилок;
- тестування (ручне або автоматизоване; програмістом, тестувальником або користувачем; в режимі чорної скриньки чи модульне…) – виявлення факту помилки;
- відтворення помилки – з'ясувати умови, за яких помилка трапляється. Це може виявитися непростим завданням при програмуванні паралельних процесів і деяких незвичайних помилках, відомих як гейзенбаги;
- налагодження – виявлення причин помилки.

6.2 Аналіз отриманих результатів

Згідно з поставленою задачею, метою даної роботи є розробка універсального цифрового ключа для систем контролю та управління доступом на основі технології RFID. В цілому розробка проекту поділяється на апаратну та програмну частини. Авжеж, таке розділення є вкрай умовним, проте для

зручності аналізу результатів скористаємось саме їм.

Апаратна частина проекту була спроєктована виходячи із специфікації стандарту RIFD. Було обрано компактний 8-бітний AVR мікроконтролер фірми Atmel, сімейства tiny, модель ATtiny85. Даний мікроконтролер було обрано через поєднання високої обчислювальної потужністі, малих габаритів, дешевизни готового пристрою та наявність восьми кілобайт енергонезалежної пам’яті. Цифрові зліпки ключів не займають багато пам’яті, а однією з цілей даної атестаційної роботи було проектування апаратної частини проекту із компактними габаритами для зручного траспортування та використання у повсякденні. Мікроконтролер серії ATtiny підходить для цієї задачі найкраще.

Як зазначалось в попередніх розділах, цифровий ключ в RFID-мітках займає лише 64 біти або 8 байт інформації, що майже не значне значення на фоні восьми кілобайт пам’яті мікроконтролера. Одночасно можно зберігати сотні різних ключів на одному девайсі, проте це вже буде несте поганий користувацький опит для кінцевого користувача.

Для написання програмної частини проекту було обрано мову C, як мову що поєднує швидкість роботи та зручність у розробці. Написана прошивка займає зовсім не багато місця в пам’яті мікроконтролера та лишає купу пам’яті для додавання цифрових ключів.

Програмну частину можна умовно поділити на розробку користувацького інтерфейсу та процес емуляції RFID-мітки. Користувацький інтерфейс у даному універсальному ключі досить примітивний, на периферійний пристрій (у якості якого виступає дисплей діагоналлю 0,96" виконаний за технологією OLED) виводиться список усіх доступних ключів, збережених у енерго-незалежній пам’яті.

Емуляція процесу спілкування ключа із RFID-считувачему було самим відповідальним етапом розробки поректу. Завдяки докладним специфікаціям стандарту ISO 14223 та теоритичним відомостім, наведених у попередніх розділах було розроблено ряд функцій для повного формування даних і

відпрвки іх за допомогою антени (або катушки індуктивності) на сторону приймаючого пристрою.

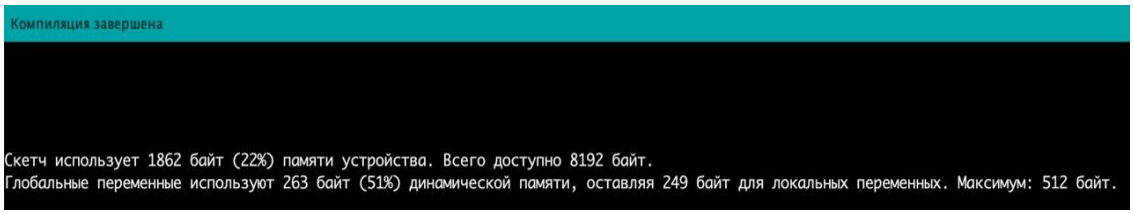

Рисунок 6.1 – Повідомлення про успішну компіляцію проекту

Після завершення написання програмної частини проекту, його було скомпільовано, на рисунку 6.1 зображено звіт середовища програмування про успішну компіляцію. Як зазначалось у попередніх розділах, пам’ять мікроконтролера ATtiny85 дорівнює 8Кб. Розроблена програма займає 1862 байти пам’яті, що приблизно дорівнює 22% від всього обсягу пам’яті мікроконтролера. Програма є досить ефективною з точки зору використання ресурсів, та має потенціал для розширення та модернізації.

В ході тестуваня програмного забезпечення було смодельовано ключ карту системи СКУД (рисунок 6.2).

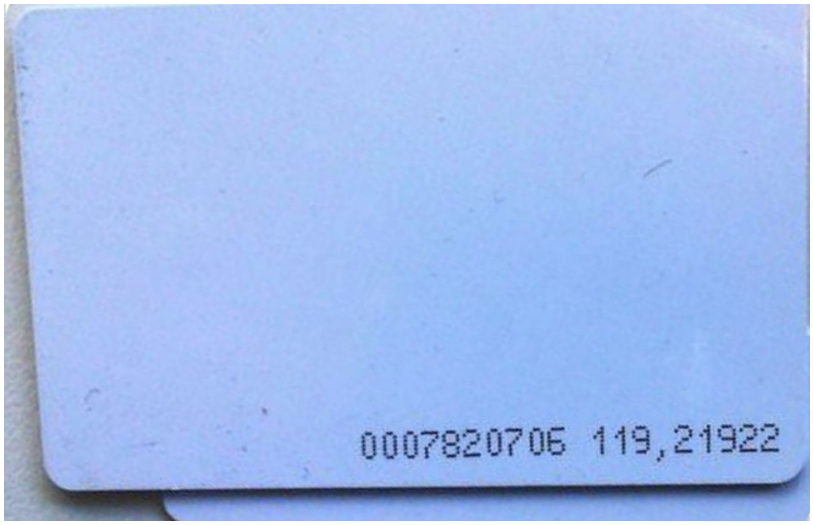

Рисунок 6.2 – RFID карта для моделювання

Добавимо цифровий ключ даної карти в програми як окрему змінну та запустимо процес емуляції. Результат процесу емуляції наведено нижче на рисунку 6.3. Як ми бачимо – результатом емуляції картки є послідовність біт, виведених у консоль.

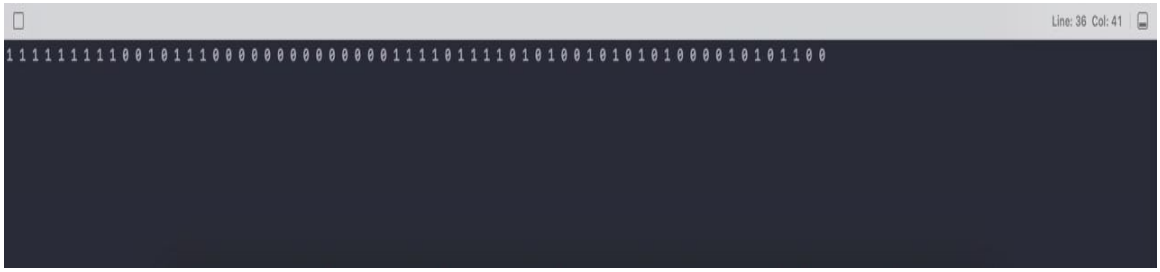

Рисунок 6.3 – Результат процесу емуляції

Розглянемо послідовність біт з рисунку 6.3 у більш зручному для сприйняття форматі. Дані згруповано у таблицю 6.1.

Таблиця 6.1 – Згруповані біти процесу емуляції

1	1	1	1	1	1	1	1	1
				0	0	1	0	1
				1	1	0	0	0
				0	0	0	0	0
				0	0	0	0	0
				0	1	1	1	1
				0	1	1	1	1
				0	1	0	1	0
				0	1	0	1	0
				1	0	1	0	0

	0	0	1	0	1
	0	1	1	0	0

Як ми бачимо, таблиця 6.1 має формат ідентичний тому, що описувався у п’ятому розділі. Вона містить перші 9 біт, які слугують заголовком та сигналізують про початок обміну даними між зчитувачем та RFID карткою. Останній стоп біт, поставлений в логічний нуль та сигналізує про кінець передачі даних. Перевіримо чи коректно було сформовано біти P0 – P9, як зазначалось у теоретичних розділах, це біти з контрольною сумою попередніх чотирьох біт, тобто рядка в таблиці. Якщо у рядку кількість нулів парна – контрольний біт буде нулем, якщо непарна – навпаки одиницею.

Тобто, якщо ми розглянемо перший рядок, то комбінація з перших чотирьох біт буде 0010. Кількість одиниць в рядку не парна, тому контрольний біт встановлено у одиницю. В другому рядку з комбінацією 1100, кількість одиниць навпаки парна, тому контрольний біт встановлено в нуль. За аналогією перевіряємо усі десять рядків, усі дані збігаються.

Залишилось перевірити лише контрольні біти PC0 – PC3. Вони формуються за аналогічно до бітів P0 – P9, за винятком того, що вони рахуються не для рядків таблиці, а для стовпчиків. Якщо взяти перший стовпчик з набором 0100000010, то в ньому парна кількість одиниць, тому контрольний біт PC0 – встановлено в нуль. В другому ж стовпчику – навпаки, комбінація 01001111100, має непарну кількість одиниць, тому контрольний біт PC1 встановлено в одиницю. Як ми бачимо дані в таблиці сформовані коректно.

Останнім кроком візьмемо біт за номерами D08 – D39, тобто безпосередній ідентифікатор клієнта, та подивимось чи коректно його було сформовано. Для цього відкинемо контрольні біти, та запишемо дані у таблицю.

Таблиця 6.2 – Отримані біти ідентифікатора клієнта

	D08;D11	D12;D15	D16;D19	D20;D23	D24;D27	D28;D31	D32;D35	D36;D39
BIN	0000	0000	0111	0111	0101	0101	1010	0010

Якщо біти ідентифікатора клаєнта перевести в десяткову систему числення, то ми отримаємо число 7820706, що повністю збігається з ідентифікатором клієнта картки, котру ми моделювали, одже процес емуляції картки пройшов вдало.

Поділ частини програми, що відповідає за емуляцію процесу спілкування з RFID-считувачем, на функціональні блоки або окремі методи значним чином полегшило процес налаштування готового коду та його тестування. Тестування окремих модулів допомогає локалізувати помилки та значноприскорює процес дебагінга.

ВИСНОВКИ

В ході розробки даної кваліфікаційної роботи було спроектовано та побудовано універсальний цифровий ключ для систем контролю та управління доступом на основі технології, на основі компактного 8-бітного AVR мікроконтролера фірми Atmel. Для реалізації даної системи було обрано мікроконтролер сімейства tiny, модель ATtiny85.

Першим чином було проаналізовано математичний апарат та було проведено аналіз технології RFID. Було обрано стандарт ISO 14223, виходячи з робочої частоти, необхідної робочої відстані, вимог до безпеки системи та ємності пам'яті.

Цифрові відбитки ключів не займають багато пам’яті, і однією з цілей даної кваліфікаційної роботи було спроектувати апаратну частину проекту з компактними розмірами для зручного транспортування та використання в повсякденному житті. Мікроконтролер серії ATtiny найкраще підходить для цього завдання.

Для апаратної частини проекту однією за найважливіших частин є котушка індуктивності, або антена. Саме завдяки їй проходить процес спілкування мікроконтролера та побудованого навколо нього цифрового ключа із RFID-зчитувачем. Для коректної роботи котушки на бажаній частоті у 125 kHz (згідно із специфікацією стандарту RFID ISO 14223) було проаналізовано математичний та фізичний апарат, котрий стоїть за індуктивніст. Було проаналізовано котушки різних форм та різні принципи побудови та обрано найоптимальніший для даного проекту варіант.

В якості периферійного пристрою в комп’ютерній системі виступає дісплейний OLED модуль з типом інтерфейсу I2C. Даний дісплейний модуль є енергоефективним та гнучким в підключенні та використанні завдяки I2C інтерфейсу.

У якості джерела вхідного сигналу в даній системі виступає контактна

кнопка, котра дає користувачеві можливість навігації списком завантажених ключів та обрання бажаного.

В ході написання проекту так було проаналізовано стандарти радіочастотної комунікації, такі як RFID, та вивчені принципи їх роботи. Проаналізовані переваги та недоліки систем побудованих навколо цієї технології.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт технічної документації та специфікації дісплейного модуля OLED-0.96-128X64 з I2C інтерфейсом [Електронний ресурс] / Режим доступу: www/ URL: https://docs.heltec.cn/download/oled/096/0.96oled-specification.pdf – 08.12.2021 р. – Загл. з екрану.

2. Сайт документації мікроконтролера ATtiny85 [Електронний ресурс] / Режим доступу: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf – 08.12.2021 р. – Загл. з екрану.

3. Шилдт, Г. Полный справочник по C [Текст] / Г. Шилдт. – СПб.: Вильямс, 2010. – 704 с.

4. Вигерс, К. И. Разработка требований к программному обеспечению. стереотипное 3-е издание [Текст] / К. И. Вигерс, Б. Джой – Спб.: БХВ-Петербург, 2017. – 736 с.

5. Баранов, В. Н. Применение Микроконтроллеров AVR: схемы алгоритмы программы 3-е издание [Текст] / В. Н. Баранов. – М. : Додэка-XXI, 2010. – 288 с.

6. Белов, А. В. Конструирование устройств на микроконтроллерах [Текст] / А. В. Белов. – М. : Наука и техника, 2009. – 256 с.

7. Finkenzeller, K. RFID Handbook [Текст] / K. Finkenzeller. – M.: Giesecke & Devrient, 2010. – 462 с.

8. Троицкий, Н. RFID-технологии на службе вашего бизнеса [Текст] / Н. Троицкий. – М.: Альпина Паблишер, 2007. – 290 с.

9. Kennedy, G. Electronic Communication Systems. 4th ed. [Текст] / G. Kennedy, B. Davis. – K.: McGraw-Hill International, 1993. – 763 с.

10. Пратта, С. Язык программирования С. Лекции и упражнения. 5-е издание [Текст] / С. Пратта. – М.: Вильямс, 2013. – 959с.

11. Гуров, В. В. Микропроцессорные системы [Текст] / В. В.Гуров. – М.: НиТ, 2019 – 336 с.

12. Матюшин, А. О. Программирование микроконтроллеров. Стратегия и тактика. Руководство [Текст] / А. О. Матюшин. – М.: ДМК, 2017. – 356 с.

13. Евстафиеев, А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL. Руководство [Текст] / А. В. Евстафиеев. – М.: ДМК, 2015. – 558с.

14. Магда, Ю. С. Современные микроконтроллеры. Архитектура, программирование, разработка устройств [Текст] / Ю. С. Магда. – М.: ДМК, 2017. – 224 с.

15. Сикорд, Р. Эффективный С. Профессиональное программирование [Текст] / Р. Сикорд. – СПб.: Вильямс, 2020. – 476 с.