Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)
магістра
(ступінь вищої освіти)

на тему **СИСТЕМА ІМІТАЦІЇ СТРІЛЕЦЬКОГО БОЮ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ РАДІОЧАСТОТНОГО ЗВ'ЯЗКУ**

Виконав: студент 2 курсу, групи КНТ-613м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Спеціалізовані комп'ютерні системи

СГАДОВ С. О
(ПРІЗВИЩЕ та ініціали)
Керівник КУДЕРМЕТОВ Р.К.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОЗІНА Г.Л.
(ПРІЗВИЩЕ та ініціали)

2024

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) «Спеціалізовані комп'ютерні системи»
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.

"____" _____ 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СГАДОВА Сергія Олександровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Система імітації стрілецького бою з використанням технології радіочастотного зв'язку
керівник проєкту (роботи) к. т. н., доцент, КУДЕРМЕТОВ Равіль Камилович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від "18" жовтня 2024 року №149
2. Строк подання студентом проєкту (роботи) 13 грудня 2023 року
3. Вихідні дані до проєкту (роботи) архітектура системи Lasertag, схеми апаратного забезпечення системи Lasertag, програмне забезпечення Lasertag
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Огляд систем LASERTAG, дослідження наявних методів організації РЧ-зв'язку;
2) Огляд методів кодування з виправленням помилок;
3) Проектування архітектури системи;
4) Проектування і розробка програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	КУДЕРМЕТОВ Р.К., доцент		
нормоконтроль	ПОЛЬСКА О.В., ст. викл.		

7. Дата видачі завдання 04.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та організації систем	20.10.2024 р.	
	імітації бою, виділення недоліків та переваг		
2	Розробка структури системи	25.10.2024 р.	
3	Розробка алгоритму роботи системи	30.10.2024 р.	
4	Проектування архітектури програмного забезпечення	15.10.2024 р.	
5	Проектування інтерфейсу користувача	20.11.2024 р.	
6	Оформлення отриманих результатів у ПЗ	29.11.2024 р.	
7	Оформлення допоміжного матеріалу	10.12.2024 р.	

Студент **Сергій СГАДОВ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Равіль КУДЕРМЕТОВ**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 90 с., 23 рис., 9 табл., 26 джерел, 2 додатка.

ІЄРАРХІЧНА СИСТЕМА, LASERTAG, ІМІТАЦІЯ БОЮ, ІЧ-ЗВ’ЯЗОК, RF-МОДУЛЬ

Об’єкт дослідження – ієрархічні радіочастотні системи.

Предмет дослідження – методи та апаратно-програмні засоби комунікації в динамічній системі.

Мета роботи – створення системи імітації бою Lasertag с централізованими засобами керування параметрами гри.

Матеріали, методи та технічні засоби: методи розробки та налагодження програмного та апаратного забезпечення мікропроцесорної системи.

Результати – розроблено та створено систему на основі модулів NRF24L01 та ІЧ-звя’зку. Для захисту від помилок обрано коди Ріда-Соломона.

Галузь використання – індустрія розваг та військові.

ABSTRACT

Explanatory note to the master's work: 90 p., 23 figures, 9 tables, 26 sources, 2 appendixes.

HIERARCHICAL SYSTEM, LASERTAG, COMBAT SIMULATION, IR COMMUNICATION, RF MODULE

The object of research is hierarchical radio frequency systems.

The subject of research is methods and hardware and software means of communication in a dynamic system.

The purpose of the work is to create a Lasertag battle simulation system with centralized way of controlling game parameters.

Materials, methods and technical means: methods of developing and debugging software and hardware of a microprocessor system.

Results - a system based on NRF24L01 modules and IR communication was developed and created. Reed-Solomon codes were chosen for error correction procedures.

Field of application is entertainment industry and military.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АЦП	аналого-цифровий перетворювач
ІЧ	інфрачервоний
РЧ	радіочастота
ЦАП	цифро-аналоговий перетворювач
ШІМ	широтно імпульсна модуляція
DBPSK	differential binary phase shift keying, диференційна фазова модуляція
EEPROM	Electrically Erasable Programmable Read-Only Memory,електрично стираєма програмована постійна пам'ять
I^2C	Inter-Integrated Circuit, послідовна асиметрична шина
GFSK	Gaussian frequency shift keying, гауссова частотна модуляція
LCD	liquid-crystal display, рідкокристалічний дисплей
LoRaWAN	low-power wide-area network для iot applications, малопотужна мережа для програм інтернету речей
MILES	Multiple Integrated Laser Engagement System, протокол багатокомпонентной інтегрованой системи лазерного боя
RS-коди	коди Ріда-Соломона
RF	radio frequency, радіочастота
SPI	Serial Peripheral Interface, послідовний периферійний інтерфейс
SRAM	static random access memory, статична оперативна пам'ять
UART	Universal asynchronous receiver/transmitter, універсальний асинхронний приймач/передав
QFSK	quadrature phase shift keying, квадратурна фазова модуляція

ВСТУП

Сучасні технології імітації зброї Lasertag відіграють важливу роль у сфері військового та цивільного навчання, забезпечуючи безпеку та ефективність тренувального процесу. Традиційні методи підготовки військовослужбовців та правоохоронних органів часто пов'язані з ризиками, пов'язаними з використанням бойової зброї. Тому необхідність створення безпечних альтернатив, які можуть відтворити реальні умови бою, стає дедалі актуальнішою. Одним із найперспективніших напрямів у цій галузі є використання інфрачервоного випромінювання для розробки систем імітації зброї. Інфрачервоні системи Lasertag дозволяють створювати високоточні симуляції, які відтворюють як механічні, і акустичні характеристики реальної зброї. Це дає можливість учасникам тренування не лише ознайомитися з тактикою застосування зброї, а й відпрацьовувати навички стрільби та командної взаємодії у безпечному середовищі. Такі системи можуть бути використані як у закритих приміщеннях, так і на відкритих майданчиках, що робить їх універсальними для різних сценаріїв.

У першому розділі наведено огляд систем Lasertag, їх різновиди, переваги та недоліки.

У другому розділі наданий огляд мікросхем для РЧ-зв’язку та їх придатність до систем Lasertag.

У третьому розділі наданий стислий огляд що до кодування з можливостю виправлення помилок для передачі інформації для імітації пострілу за допомогою ІЧ-сигналу.

У четвертому розділі виконується аналіз вимог до системи та розроблено апаратне забезпечення для моделі зброї, датчику влучень та керуючій базової станції.

У п’ятому розділі виконується розробка програмного забезпечення системи, об’єктної моделі ПО зброї, систему кодування та розкодування

повідомлень, інтерфейс керування грою, а також розроблена система відтворення звукових ефектів в асинхронному режимі. Також наведені результати випробувань системи.

Основною метою даної роботи є попередній аналіз компонент важливих для розробці системи імітації зброї на основі інфрачервоного випромінювання с використанням кодів корекції помилок та керування параметрами бою з допомогою технології РЧ-зв’язку, та розробка архітектури й програмного забезпечення системи імітації бою.

Для цього потрібно вирішити такі завдання: розробити ієрархічну систему на основі радіозв'язку, розробити мікропроцесорне обладнання для кінцевих точок системи, розробити алгоритми роботи цих пристроїв, а саме моделей автоматичних гвинтівок, датчиків, базової станції, що управляє системою.

Апробація отриманих результатів надана в [1-4].

1 ОГЛЯД СИСТЕМ LASERTAG

1.1 Історія систем LASERTAG

Lasertag – це командна військово-тактична гра з використанням безпечної лазерної зброї та датчиків влучення. Датчики кріпляться на жилетах, шоломах та нарукавниках. Лазерна зброя може бути сконструйована з удароміцного пластику, реальної ваги (масогабаритний-макет), страйкбольних приводів або мисливської зброї. Ігри в Lasertag проводяться на природі, у торгових центрах, на добре обладнаних стадіонах та стрільбищах, а також у закритих приміщеннях.

Променевий тренажер, що використовується як навчальний посібник для тактичної та вогневої підготовки у військових підрозділах США, був розроблений на початку 1960-х років [5]. Необхідність в такому обладнанні виникла не тільки для проведення занять з бойової підготовки, але й для підвищення рівня колективної підготовки військ та контролю результатів військових навчань (коли для цих цілей використовувалися холості патрони, офіцери-спостерігачі та відповідальні за підбиття підсумків навчань не могли зафіксувати перевагу одних з інших). Один з таких тренажерів для піхоти, з записуючим пристроєм і звуковим сигналом при попаданні, був розроблений у військовій лабораторії годинникової компанії Bulova в Квінсі (хоча у 1959 році увагу військового керівництва привернула далекобійна версія оригіналу, розроблена для військових) [6]. Замість лазерів використовувався невидиме людському оку інфрачервоне світло, що тільки підвищувало реалістичність використовуваної системи імітації бою. Індивідуальні комплекти складалися з генератора інфрачервоного сигналу, що прикріплюється до окремих видів зброї (пістолетів, штурмових гвинтівок і кулеметів), а також джерела живлення та записуючого пристрою в контейнері розміром з флягу, що прикріплюється до поясного ременя (див. рис. 1.1) [6,7].

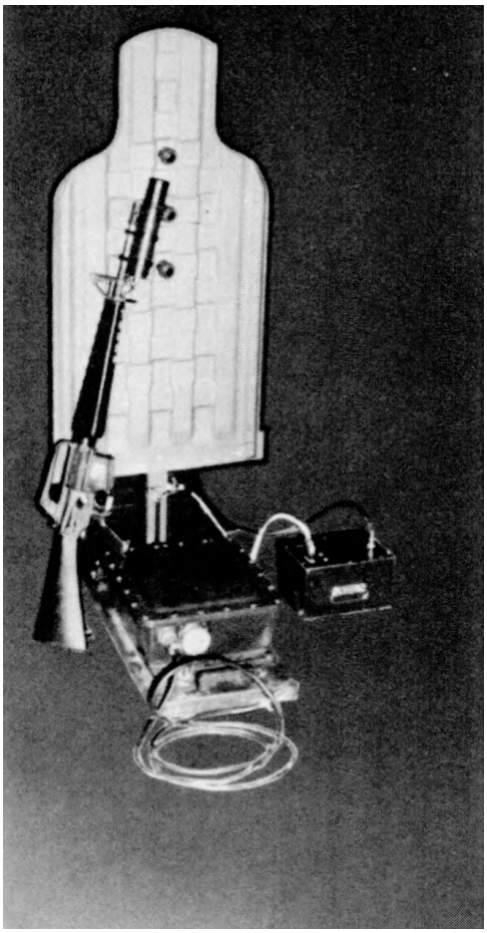

Рисунок 1.1 – Гвинтівка M16A1 з лазерним наведенням (1972)

Історія комерційної ідеї управління лазертагом почалася 1977 року. Пізніше Джордж Картер III зі США вигадав гру «Фотон» після перегляду фільму «Зоряні війни». Саме його ідея лягла в основу всіх сучасних комерційних лазертагів як за стилем проведення, так і за концепцією гри.

У 1979 році компанія South Bend Toys представила гру Star Trek Phasers. Гра була набір з двох пістолетів (бластерів) з прикріпленими до них датчиками [8]. Гра була одночасно простою і складною, але все ж таки мала одну відмінну особливість - в ній була можливість відображати повторювані близькі постріли і рикошети в милях. Попадання та промахи відстежувалися залежно від рівня потужності випромінювача (у 100% сучасних систем ймовірність промаху розраховується програмно). У практичному плані продаж цієї гри почався в 1986 році. Термін " Lasertag" був спочатку введений компанією Worlds-of-Wonder у 1988-89 роках (яка виробляла обладнання споживчого класу), а потім, з

невеликими змінами (заміна "z" на "s"), для всіх систем, що використовують фотонний принцип.

Lasertag та аналогічні принципи використовуються в армійській системі імітації вогню та поразки, яка з'явилася в армії США наприкінці 1970-х років як спосіб імітації бойових дій, навчання та готовності солдатів. Для цього було розроблено систему MILES (Multiple Integrated Laser Engagement System). Вона складалася із спеціальних випромінювачів, прикріплених до бойової зброї. Сигнали випромінювачів уловлювалися датчиками, вбудованими у форму кожного учасника [9]. Після досить тривалого використання (до початку 1990-х років) система MILES була замінена її вдосконаленою версією - MILES-2000, яка чудово показала себе у Форт-Беннінгу у 1998 році. В даний час військові системи імітації стрільби та бою випускаються виробниками США та країн НАТО.

1.2 Типи Lasertag

Лазерні системи імітації тактичного бою є основним засобом профільної підготовки військовослужбовців у країнах НАТО.

Arena Lasertag – гра проходить у закритому приміщенні у стаціонарному лабіринті. Перешкоди на арені, стіни, підлога та ігрові об'єкти підсвічуються флуоресцентною фарбою. Команди зазвичай складаються з п'яти чи шести гравців. Наприклад, два гравці – «атакуючі», два – «захисники» і один – «мисливець» (нападник). Стандартна гра триває 30 хвилин.

Extra Arena Lasertag – призначена для гри на відкритому повітрі та проводиться на спеціально обладнаних тренувальних полях, у парках чи лісах. Кількість гравців та розподіл ролей практично не обмежена і визначається лише завданнями гри та умовами місцевості. Середня тривалість бою становить від 20 хвилин до години, залежно від завдання, можливі обмеження часу.

Military Lasertag – повномасштабна симуляція тактичного бою.

Mobile Lasertag – лазертаг зі спеціальними надувними фігурами, які імітують перешкоди під час гри. Може проводитись на вулиці, у спортзалах та інших приміщеннях.

Sport Lasertag – напрямок, в якому змагаються члени регіональних відділень, при цьому передбачені змагання та турніри, а результати підбиваються у загальній статистиці регіонів та гравців.

Зомбітаг-змагання з лазертагу зі сценаріями, в яких гравці однієї команди переходять на бік супротивника та грають.

Лазертаг-біатлон – тренування з гвинтівками та електронними мішенями.

Домашній лазертаг (або «сімейний лазертаг») – орієнтований на сегмент «споживчого» класу – призначений для придбання обладнання для гри в приміщенні та в селі, не для комерційного використання.

Танковий лазертаг – лазерне обладнання встановлюється на танки, іграшкові всюдиходи та позашляховики. Існує також «гібридний» напрямок – firetag, що є мисливською зброєю з лазертагівською електронікою і корпусом, що катапультується.

1.3 Висновки

У порівнянні з пейнтболом та страйкболом лазертаг має суттєві переваги:

- відсутність витратних матеріалів (боєприпасів та фарби);
- не схильні до впливу вітру при стрільбі;
- велика дальність дії;
- немає необхідності одягати захисний одяг, шоломи чи маски;
- немає болю чи травм, якщо «постріл» потрапляє у гравця;
- попадання записуються в електронному вигляді, що знижує ризик нечесної гри з боку однієї з команд (наприклад, якщо суперник не розпізнає попадання).

Гра має й недоліки. Наприклад, потрапляючи в гравця з відстані у матчі на додатковій арені, стрілець не бачить і не відчуває, що влучив у ціль. Постріли супроводжуються звуковими ефектами (у військових модифікаціях для звукової імітації бою використовуються холості патрони). Тому необхідно щоб гвинтівки Lasertag були з’єднані у єдину мережу, але були також відносно мобільні. Також необхідна якісна імітація відчуття реальної зброї.

2 ПОРІВНЯЛЬНИЙ АНАЛІЗ МІКРОСХЕМ ДЛЯ БЕЗДРОТОВОГО ЗВ'ЯЗКУ

2.1 Огляд популярних РЧ модулів

Розглянемо п'ять різних мікросхем, призначених для використання у системах бездротового зв'язку [8-10]: NRF24L01, ESP8266, ESP32, CC1101 та SX1278. Проаналізуємо їх особливості, застосування та обмеження.

Мікросхеми NRF24L01, ESP8266 та SX1278 можуть бути використані з мікроконтролером STM32F405 для різних програм.

У статті [8] дослідники описують застосування бездротових мереж датчиків з використанням протоколу LoRaWAN та Wi-Fi для різних програм. Автори підкреслюють важливість використання відповідного протоколу передачі в залежності від вимог до продуктивності та дальності зв'язку.

Використання LoRaWAN та Wi-Fi дозволяє створювати бездротові мережі датчиків, які можуть забезпечувати точне визначення розташування об'єктів. У статті згадується застосування мікроконтролера STM32F4 (аналогічно STM32F405) з лазерним датчиком для забезпечення зв'язку.

В іншій статті [9] дослідник описує застосування мікроконтролера ESP8266 з використанням протоколу Wi-Fi для різних програм. Автор підкреслює важливість використання відповідного мікроконтролера залежно від вимог до продуктивності та дальності зв'язку.

Використання мікроконтролера ESP8266 дозволяє створювати системи, які можуть забезпечувати точне визначення розташування об'єктів. У статті згадується застосування лазерного датчика для забезпечення точного розташування об'єктів.

Стислі характеристики мікросхем для бездротового зв'язку подані у табл. 2.1.

Таблиця 2.1 - Характеристики мікросхем для бездротового зв'язку

Характеристика	**NRF24L01**	**ESP8266**	**ESP32**	**CC1101**	**SX1278**
Частотний діапазон	2.4 ГГц	2.4 ГГц	2.4 ГГц, 5 ГГц	Налаштований	868/915 МГц
Модуляція	GFSK	DBPSK, DQPSK, DPSK	DBPSK, DQPSK, DPSK, QPSK	Настроювана	LoRa
Протоколи	Власний	Wi-Fi, TCP/IP	Wi-Fi, Bluetooth, TCP/IP	Власний	LoRaWAN
Обчислювальна потужність	Ні	Вбудований мікроконтролер	Вбудований двоядерний мікроконтролер	Ні	Ні
Периферія	SPI	SPI, UART, ADC, GPIO	SPI, UART, ADC, GPIO, I2C, I2S, SDIO та ін.	SPI, UART	SPI, UART
Ціна	Низька	Середня	Середня-висока	Середня	Середня

NRF24L01 – це класичний і широко використовується RF-модуль, що працює в діапазоні 2.4 ГГц. Він має гарне співвідношення ціни та якості, що зробило його популярним вибором для багатьох аматорських та комерційних проектів. Однак, з розвитком технологій з'явилися й інші модулі, що пропонують ширші можливості та характеристики.

ESP8266 і **ESP32** – це популярні Wi-Fi модулі, які, крім радіочастотного зв'язку, пропонують вбудований мікроконтролер і підтримку різних протоколів,

включаючи TCP/IP. Вони ідеально підходять для створення складніших IoT-пристроїв.

ESP8266 має такі характеристики:

– напруга живлення 3.3V (2.5-3.6V);

– струм споживання 300 мА при запуску та передачі даних, 35 мА під час роботи, 80 мА в режимі точки доступу;

– максимальний струм на ніжку – 12 мА;

– Flash пам'ять (пам'ять програми) 1 МБ;

– Flash пам'ять (файлове сховище) 1-16 МБ залежно від модифікації;

– EEPROM пам'ять до 4 кБ;

– SRAM пам'ять 82 кБ;

– частота ядра 80/160 МГц;

– 11 біт портів;

– 10 портів ШІМ;

– 10 переривань;

– WiFi зв'язок.

Мікроконтролер та управління:

– Tensilica Xtensa LX6 двоядерний (або одноядерний) 32-розрядний процесор з тактовою частотою 160 або 240 МГц і продуктивністю до 600 DMIPS (Dhrystone MIPS);

– співпроцесор із ультранизким енергоспоживанням;

– має пам'ять 520 КБ пам'яті SRAM.

Бездротовий зв'язок:

– Wi-Fi 802.11 b/g/N;

– Bluetooth v4.2.

Периферійні інтерфейси:

– 12-розрядний АЦП до 18 каналів;

– 2 × 8 біт ЦАПа;

– 10 × портів для підключення ємнісних датчиків;

– датчик температури;

– SPI, I²C, UART інтерфейси;

– Ethernet MAC interface з виділеним DMA та IEEE 1588 Precision Time.

Підтримка протоколів:

– ІЧ дистанційне керування (передавач/приймач, до 8 каналів);

– можливість підключення двигунів та світлодіодів через ШІМ –вихід;

– датчик Хола;

– аналоговий підсилювач низького енергоспоживання.

Таким чином**, CC1101** – це універсальний модуль, який може працювати у різних частотних діапазонах та підтримує різні протоколи. Він має високу гнучкість налаштування і дозволяє створювати кастомні радіосистеми.

Області застосування трансівера CC1101:

– у промисловому моніторингу та управлінні;

– у малопотужних додатках бездротового зв'язку, які працюють у таких діапазонах частот: ism та srd, 315, 433, 868 та 915мгц;

– у системах бездротової сигналізації;

– у мережах бездротових датчиків;

– у домашній чи будівельній автоматизації;

– в автоматичному зчитуванні показань із вимірювальних приладів.

Є таки риси:

– у режимі сну споживання струм не перевищує 400нА;

– швидкий запуск, у режим передачі або прийому з режиму сну трансівер переходити за рекордні 240мкс;

– можливість поновити роботу при виявленні достатнього радіочастотного сигналу;

– буфери розділені, 64-бітні, за рахунок цього дозволяють передавати дані потоками.

Порівняємо NRF24L01 з деякими іншими популярними модулями, враховуючи такі параметри, як дальність зв'язку, швидкість передачі даних, енергоспоживання, вартість та область та застосування (див. табл. 2.2).

Таблиця 2.2 - Порівняння мікросхем для бездротового зв'язку

Характеристика	NRF24L01	ESP8266	ESP32	CC1101	SX1278
Дальність зв'язку	Середня (десятки метрів)	Середня-висока (до кількох сотень метрів)	Середня-висока (до кількох сотень метрів)	Середня (десятки-сотні метрів)	Довга (кілометри)
Швидкість передачі	Середня (Мбіт/с)	Висока (Мбіт/с)	Дуже висока (Мбіт/с)	Середня (Мбіт/с)	Низька (кбіт/с)
Енергоспоживання	Низьке	Середнє	Середнє	Низьке	Дуже низька
Протоколи	Власний	Wi-Fi, TCP/IP	Wi-Fi, Bluetooth, TCP/IP	Власний	LoRaWAN
Інтеграція	Мікроконтролер	Мікроконтролер Wi-Fi	Мікроконтролер Wi-Fi, Bluetooth	Мікроконтролер	Мікроконтролер
Вартість	Низька	Середня	Середня	Середня	Середня
Складність використання	Середня	Середня-висока	Середня-висока	Середня	Середня
Типові застосування	Бездротові мережі, радіокерування, IoT	IoT,домашня автоматизація, бездротові мережі	IoT,домашня автоматизація, пристрої, що носяться	Бездротові мережі, радіокерування	IoT, промисловий моніторинг

2.2 Висновки

SX1278 – це модуль, що працює за протоколом LoRaWAN, що забезпечує далекий зв'язок із низьким енергоспоживанням. Він ідеально підходить для створення великомасштабних мереж IoT.

Спеціалізована мікросхема NRF24L01 призначена для бездротового зв'язку в діапазоні 2.4 ГГц. Використовує цифрову модуляцію GFSK, що забезпечує хорошу стійкість до перешкод. Простота використання завдяки

широкій підтримці різними мікроконтролерами є однією з її головних особливостей.

Низьке енергоспоживання в режимах сну та прийому дозволяє використовувати цю мікросхему в різних пристроях, включаючи бездротові сенсорні мережі. Можливість роботи в мережі з кількома пристроями робить її ідеальною для використання у радіокерованих моделях.

Застосування цієї мікросхеми можна знайти у простих IoT-проектах та передачі невеликих обсягів даних на короткі відстані. Однак обмежена дальність зв'язку в порівнянні з деякими іншими модулями є значним недоліком. Невисока швидкість передачі даних також не дозволяє використовувати її у додатках, що вимагають високих швидкостей.

З іншого боку, мікросхема ESP8266 є системою на кристалі (SoC) з вбудованим 32-бітовим процесором Tensilica Xtensa LX106 і підтримкою TCP/IP стека. Висока обчислювальна потужність для виконання складних завдань робить її ідеальною для використання у IoT-пристроях.

Гнучкість налаштування завдяки відкритому вихідному коду прошивки також дозволяє використовувати цю мікросхему в різних додатках, включаючи бездротові мережі та розробку протоколів користувача. Можливість роботи як точка доступу або клієнта робить її ще більш привабливою.

Однак відносно високе енергоспоживання в активному режимі є значним недоліком цієї мікросхеми. Можуть знадобитися додаткові знання з програмування для ефективного використання ESP8266.

Мікросхема ESP32 є двохдерним 32-бітовим процесором Xtensa LX6 з вбудованими модулями Wi-Fi і Bluetooth. Висока продуктивність і багатозадачність роблять її ідеальною для використання в різних додатках, включаючи пристрої, що носяться.

Підтримка різних протоколів, включаючи BLE, MQTT і HTTP, також дозволяє використовувати цю мікросхему в широкому спектрі додатків. Можливість роботи в режимі енергозбереження робить її ще більш привабливою для використання в IoT-пристроях із високими вимогами до продуктивності.

Однак більш висока вартість порівняно з ESP8266 є значним недоліком цієї мікросхеми. Для ефективного використання ESP32 можуть знадобитися додаткові знання з програмування.

Спеціалізована мікросхема CC1101 призначена для радіочастотного зв'язку та підтримує різні частотні діапазони та модуляції. Висока стійкість до перешкод робить її ідеальною для використання в бездротових сенсорних мережах.

Застосування цієї мікросхеми можна знайти у простих IoT-проектах та передачі невеликих обсягів даних на короткі відстані. Проте, низька швидкість передачі є значним недоліком. Відсутність вбудованого мікроконтролера також робить її менш привабливою для використання у складних програмах.

З іншого боку, спеціалізована мікросхема SX1278 призначена для радіочастотного зв'язку з довгим ходом та підтримує різні частотні діапазони та модуляції. Висока стійкість до перешкод робить її ідеальною для використання в великомасштабних IoT-мережах.

Застосування цієї мікросхеми можна знайти у сільському господарстві, моніторингу навколишнього середовища та промислових додатках. Довга дальність зв'язку в умовах міської забудови робить її ще привабливішою для використання у складних мережевих системах.

Проте, низька швидкість передачі є значним недоліком цієї мікросхеми. Обмежена пропускна здатність також не дозволяє використовувати її у додатках з високими вимогами до продуктивності.

ESP32 ідеальна для використання в пристроях та IoT-пристроях з високими вимогами до продуктивності, але має більш високу вартість [10]. Спеціалізована мікросхема CC1101 призначена для радіочастотного зв'язку та підтримує різні частотні діапазони та модуляції, але обмежена низькою швидкістю передачі даних.

CX1278 ідеальна для використання у великомасштабних IoT-мережах, але має обмежену пропускну здатність. У кожному випадку вибір мікросхеми залежить від конкретної програми та вимог до продуктивності.

Використання системи LASERTAG дозволяє створювати мережі, які можуть забезпечувати точне визначення розташування об'єктів. У статті згадується застосування мікроконтролера STM32F4 (аналогічно STM32F405) з лазерним датчиком для забезпечення зв'язку об'єктів.

Насамкінець, кожна з цих мікросхем має свої сильні та слабкі сторони. NRF24L01 ідеальна для простих IoT-проектів та радіокерування моделями, але обмежена дальністю зв'язку та швидкістю передачі даних. ESP8266 є потужним інструментом для розробки складних мережевих систем, але вимагає додаткового знання з програмування.

Загалом, усе наголошує на важливості використання відповідного протоколу передачі даних та алгоритму обробки сигналів залежно від вимог до продуктивності та дальності зв'язку. Застосування мікроконтролера STM32F4 (аналогічно STM32F405) з ІЧ датчиком для забезпечення надійного зв'язку об'єктів є загальною темою.

3 МЕТОДИ КОДУВАННЯ З ВИПРАВЛЕННЯМ ПОМИЛОК

3.1 Огляд методів завадостійкого кодування

Класичні методи завадостійкого кодування заклали фундамент у розвиток теорії кодування. Кожен із цих методів має свої переваги та недоліки, і вибір оптимального методу залежить від конкретних вимог системи зв'язку. У сучасних системах зв'язку все частіше використовуються складніші методи, такі як LDPC-коди та турбокоди, які забезпечують більш високу продуктивність. Однак, класичні методи, як і раніше, широко застосовуються завдяки своїй простоті та ефективності в певних умовах.

Приведемо огляд деяких класичних методів кодування з виправленням помилок.

3.1.1 Коди Хемінгу

Коди Хемінга [11,12] будуються з урахуванням матриці перевірки на парність H, що має розмірність (mr)×n, де n – довжина кодового слова, m – число інформаційних біт, r – число перевірочних біт. Матриця H повинна відповідати певним умовам, щоб забезпечити виявлення та виправлення одиночних помилок. При прийомі кодового слова обчислюється синдром $s = HrT$, де r отриманий вектор. Якщо синдром дорівнює нулю, то помилок немає. Якщо синдром не дорівнює нулю, він вказує на позицію помилкового біта, який необхідно виправити. Перевагами є простота реалізації, висока ефективність виявлення та виправлення одиночних помилок. Їх недоліки це - обмежена здатність до виправлення багатьох помилок, низька швидкість передачі даних при збільшенні числа інформаційних біт.

3.1.2 Циклічні коди

Циклічні коди [13] є підмножиною лінійних блокових кодів, що володіють властивістю циклічності. Вони можуть бути описані за допомогою генераторного полінома g(x). Для декодування циклічних кодів використовується синдромне декодування. Синдром обчислюється шляхом розподілу отриманого полінома на генераторний поліном. Залишок від поділу і є синдромом.

3.1.3 Коди BCH та Ріда-Соломона

Коди BCH [14] є узагальненням циклічних кодів і мають більш високу здатність до виправлення помилок. Коди Ріда-Соломона є підклас BCH-кодів і широко використовуються в системах зберігання даних. Перевагами є висока ефективність для виправлення помилок, регулярна структура, що спрощує апаратну реалізацію. Їх недоліки це - складність побудови кодів із заданими параметрами, відносно висока складність алгоритмів декодування.

3.1.4 Згорткові коди

Згорткові коди [15] кодують інформацію не блоками, а безперервним потоком даних. Вони реалізуються за допомогою регістрів зсуву із зворотним зв'язком.

3.1.5 Алгоритм Вітербі

Алгоритм Вітербі є найефективнішим алгоритмом декодування згорткових кодів. Він заснований на принципі динамічного програмування і дозволяє знайти найімовірнішу послідовність переходів по ґратах станів кодера. Перевагами є висока стійкість до перешкод, гнучкість у виборі параметрів коду. Їх недоліки це– висока обчислювальна складність алгоритму декодування, особливо довгих обмежень коду.

Таблиця 3.1 - Порівняння класичних методів

Характеристика	**Коди Хеммінгу**	**Циклічні коди**	**Згорткові коди**
Структура	Блокові	Блокові	Згорткові
Здатність до виправлення помилок	Поодинокі помилки	Численні помилки	Численні помилки
Складність реалізації	Низька	Середня	Висока
Швидкість декодування	Висока	Середня	Низька
Гнучкість	Низька	Середня	Висока

3.1.6 LDPC-коди (коди з низькою щільністю перевірок на парність)

LDPC-коди [16,17] є класом лінійних блокових кодів, що характеризуються матрицею перевірки на парність з низькою щільністю одиниць. Це означає, що у матриці перевірки більшість елементів дорівнюють нулю.

LDPC-коди можуть бути представлені у вигляді біпартитного графа, де вершини одного типу відповідають змінним (інформаційним бітам), а вершини іншого типу – перевіркам. Ребра з'єднують змінні та перевірки, що відповідають ненульовим елементам матриці перевірки. Для декодування LDPC-кодів використовують ітеративні алгоритми, такі як алгоритм поширення вірогідностей (belief propagation). Ці алгоритми дозволяють досягати високої продуктивності за відносно низької складності. Перевагами є висока стійкість до перешкод, гнучкість у виборі параметрів коду, можливість досягнення близьких

до теоретичної межі значень швидкості передачі даних. Їх недоліки це – висока складність побудови оптимальних кодів, необхідність великих обчислювальних ресурсів для декодування.

3.1.7 Турбокоди

Турбокоди [18,19] є конкатенацією двох або більше згорткових кодів, розділених інтерлівером. Інтерлівер служить для перемішування біт кодового слова, що дозволяє знизити кореляцію між помилками та підвищити ефективність декодування. Декодування турбокодів здійснюється ітеративно. На кожній ітерації декодери складових кодів обмінюються м'якими інформаційними бітами, що дозволяє покращити оцінку ймовірностей. Перевагами є висока стійкість до перешкод, близька до межі Шеннона. Їх недоліки це – висока обчислювальна складність, потреба у великій кількості ітерацій для досягнення збіжності.

3.1.8 Полярні коди

Полярні коди [20] є відносно новим класом кодів, розробленим Еріком Поляром у 2009 році. Вони мають низку унікальних властивостей і теоретично досягають межі Шеннона для симетричних каналів без пам'яті. Полярні коди будуються з урахуванням рекурсивного процесу поляризації каналу.

Для декодування полярних кодів використовуються субоптимальні алгоритми, які дозволяють досягти високої продуктивності за відносно низької складності. Перевагами є висока продуктивність, проста апаратна реалізація. Їх недоліки це складність побудови кодів великих довжин блоків.

Таблиця 3.2 – Порівняння сучасних методів кодування

Характеристика	LDPC-коди	Турбокоди	Полярні коди
Структура	Блокові	Конкатенація згорткових кодів	Рекурсивна побудова
Алгоритми декодування	Ітеративні (BP)	Ітеративні	Субоптимальні
Перешкодостійкість	Висока	Дуже висока	Висока (близька до межі Шеннона)
Складність реалізації	Середня	Висока	Середня
Гнучкість	Висока	Середня	Висока

3.2 Висновки

Можна очікувати, що коди Ріда-Соломона або LDPC більш цікави що до використання у системах Lasertag. Але незважаючи на високу ефективність виправлення помилок, LDPC коди забезпечують відмінні характеристики виправлення помилок, особливо при високих рівнях шумів і близькість до теоретичних меж: LDPC коди близькі до межі Шеннона, що означає, що вони можуть досягати високої пропускної здатності при заданому рівні помилки, LDPC коди складніші у реалізації (ітеративні алгоритми).

Коди Ріда-Соломона (RS-коди) є потужним інструментом для корекції помилок, що робить їх особливо корисними в системах передачі даних, таких як LaserTag. Коди Ріда-Соломона здатні виправляти помилки, що виникають під час передачі даних через ІЧ-сигнали. В умовах, де можливі перешкоди або загасання сигналу, такі як ігри, LaserTag, використання RS-кодів дозволяє значно підвищити надійність передачі інформації, забезпечуючи коректне отримання даних навіть за наявності помилок. RS-коди дозволяють ефективно використовувати доступну смугу пропускання, мінімізуючи кількість повторних передач. Це особливо важливо у динамічних ігрових ситуаціях, де швидкість реакції має значення. За допомогою RS-кодів можна передавати більше інформації за менший час, що покращує загальну продуктивність системи. Коди Ріда-Соломона відносно легко реалізуються на рівні програмного забезпечення та апаратного забезпечення. Наприклад, для систем на базі Arduino, що використовуються в LaserTag, існують готові бібліотеки та приклади коду для роботи з цими кодами, що спрощує розробку та інтеграцію до існуючих систем.

RS-коди можуть бути адаптовані для роботи з різними форматами даних, що робить їх універсальними для застосування у різних ігрових режимах та сценаріях. Це дозволяє розробникам легко налаштовувати систему під конкретні вимоги та умови гри. ІЧ-сигнали схильні до впливу зовнішніх факторів, таких як світлові перешкоди або інші джерела інфрачервоного випромінювання. RS-коди

допомагають мінімізувати вплив цих шумів на якість передачі, забезпечуючи більш стабільну роботу системи в різних умовах. Використання кодів Ріда-Соломона у системах LaserTag значно покращує якість передачі даних та надійність роботи обладнання, що є критично важливим для успішної гри.

Lasertag як гра має й недоліки. Наприклад, потрапляючи в гравця з відстані у матчі на додатковій арені, стрілець не бачить і не відчуває, що влучив у ціль. Постріли супроводжуються звуковими ефектами (у військових модифікаціях для звукової імітації бою використовуються холості патрони). Тому необхідно щоб гвинтівки Lasertag були з’єднані у єдину мережу, але були також відносно мобільні. Стрілець має отримувати інформацію про влучання. Також необхідна якісна імітація відчуття реальної зброї.

РЧ-модуль на базі NRF24L01 добре підходить для простих IoT-проектів та радіокерування моделями, але обмежена дальністю зв'язку та швидкістю передачі даних. Але його достатньо для дистанцій до кілометра, що досить для гри. Крім того якщо робити зв’язок «гвинтівка-датчик» це більш ніж треба. Якщо отримувати інформацію про влучання «гвинтівка-гвинтівка», то й не треба очікувати дистанцій більш за ста метрів.

Коди Ріда-Соломона (RS-коди) є потужним інструментом для корекції помилок, що робить їх особливо корисними в системах передачі даних, таких як ІЧ -проміні. Коди Ріда-Соломона здатні виправляти помилки, що виникають під час передачі даних через ІЧ-сигнали. В умовах, де можливі перешкоди або загасання сигналу, такі як ігри LaserTag, використання RS-кодів дозволяє значно підвищити надійність передачі інформації, забезпечуючи коректне отримання даних навіть за наявності помилок. RS-коди дозволяють ефективно використовувати доступну смугу пропускання, мінімізуючи кількість повторних передач. Це особливо важливо у динамічних ігрових ситуаціях, де швидкість реакції має значення. За допомогою RS-кодів можна передавати більше інформації за менший час, що покращує загальну продуктивність системи. Коди Ріда-Соломона відносно легко реалізуються на рівні програмного забезпечення та апаратного забезпечення. Для систем на базі STM32, що використовуються в

LaserTag, існують готові бібліотеки та приклади коду для роботи з цими кодами, що спрощує розробку та інтеграцію до існуючих систем.

RS-коди можуть бути адаптовані для роботи з різними форматами даних, що робить їх універсальними для застосування у різних ігрових режимах та сценаріях. ІЧ-сигнали схильні до впливу зовнішніх факторів, таких як світлові перешкоди або інші джерела інфрачервоного випромінювання. RS-коди допомагають мінімізувати вплив цих шумів на якість передачі, забезпечуючи більш стабільну роботу системи в різних умовах. Використання кодів Ріда-Соломона у системах LaserTag значно покращує якість передачі даних та надійність роботи обладнання, що є критично важливим для успішної гри

Загалом, усе наголошує на важливості використання відповідного протоколу передачі даних та алгоритму обробки сигналів залежно від вимог до продуктивності та дальності зв'язку. Застосування мікроконтролера STM32F4 (аналогічно STM32F405) з ІЧ датчиком для забезпечення надійного зв'язку об'єктів є гідною темою в купе з РЧ-модулем на базі NRF24L01 та кодами Ріда-Соломона.

У ході подальшої роботи слід розробити систему LaserTag , яка складається з імітаційних моделей гвинтівок, налобних ІЧ-датчиків влучення та керуючої (базової) станції. Для організації управлінням імітації бою ці компоненти мають бути пов'язані у бездротову мережу. Кожна гвинтівка має як індикувати локальні параметри гравця (здоров'я, потрапляння, кількість набоїв), так і вміти передавати статистику на базову станцію. Також важливо здійснювати світлозвуковий супровід бою. Тому при проектуванні системи необхідно вирішити , як мінімум, такі завдання:

- здійснення РЧ - зв'язку;
- реалізацію модуляції ІЧ-сигналу;
- реалізацію прийому ІЧ-сигналу;
- реалізацію алгоритмів завадостійкого кодування ІЧ-сигналу;
- керування LCD- екраном;
- відтворення звукових ефектів.

4 РОЗРОБКА АПАРАТНОГО ЗАБЕСПЕЧЕННЯ

4.1 Визначення та аналіз вимог до апаратного забезпечення

Розглянемо вимоги, яким повинна відповідати система, що розробляється з точки зору користувача.

Логістика гри. Підготовка до запуску. Перед боєм кожна зброя має одержати номер (від 1 до 100), який відповідає номеру каналу. Це здійснюється через ІЧ-світлодіод або РЧ-канал командного пульта. Для цього при включенні зброї утримується кнопка затвора після чого зброя переходить в сервісний режим на 10 секунд. Далі воно переходить у режим очікування.

Датчику влучення також одноразово задається номер, який пов'язує його зі зброєю. Це відбувається за допомогою ІЧ-пакету прив'язки, який може бути сприйнятий датчиком попадання тільки в сервісному режимі, в який він входить на кілька секунд після включення. Тобто, та рушниця, яка вистрілить у нього, в цей момент дасте йому свій номер. Номери каналів запам'ятовуються зброєю та датчиками в незалежній пам'яті і стираються при вимкненні.

Перше, що відбувається в режимі очікування, спроба зв'язатися з датчиком попадання на своєму каналі по РЧ. Якщо спроба не вдалася, зброя блокується і при натисканні на курок ІЧ-світлодіод видає пакет прив'язки датчика, який повинен потрапити на датчик влучення.

Таким чином, попередньо налаштовані пов'язки-датчики і зброя лунають граючим. Після цього командний пульт роздає по всіх каналах пакети конфігурації гри (наприклад, кількість набоїв, кількість пунктів здоров'я, тривалість бою), складаючи при цьому список учасників, тобто включених рушниць. Після чого віддається широкомовна команда «Старт», стирається внутрішній журнал бою та гвинтівки переходять у режим бою.

Початок гри. Старт бою надсилається в радіо режимі. За командою з пульта СТАРТ гвинтівка видає звук та з'являється можливість стрілянини. При влученні у гравця забирають життя рівну силі пострілу. Якщо життів залишилося

понад 0, це поранення. При цьому активується шоковий стан (тривалість його задається з пульта): вібрація з перериваннями, звук поранення, миготіння датчика, на екрані гвинтівки № стовбура, хто поранив. у шоковому стані гвинтівка не приймає сигналів ураження та не може стріляти. Гвинтівка після поранення відсилає радіо сигнал гвинтівці.

При повному ураженні (життя 0 і менше) гвинтівка відсилає сигнал гравцеві , що вразив про поразку та виведення з гри гравця окремий звук. типу – «Ворог ліквідований!» Активується вібрація, яка відмінна від поранення більш тривала з переривання приблизно п’ять секунд, та мерехтливе з інтервалом підсвічування. А також звук, що повідомляє про поразку. Після переходить в режим очікування. Після бою, у уражених гравців і ті гравці яких командою “стоп” перевели в режим очікування відсилають пульту-базі статистику . бою скине два звіти. стоп-бою. Також знадобиться регулювання сили ІЧ випромінювання.

Екран у режимі бою повинен відображати, накштальт, як у [21] (див. рис. 4.1):

– індикатор підключення датчика ураження та номер ствола (гвинтівки);
– індикатор заряду батареї;
– індикатор заряду датчика;
– кількість життів – до 100;
– кількість обойм;
– кількість заряджених патронів;
– отримані поранення у цьому бою;
– хто поранив останній;
– кількість влучень.

Після закінчення бою за часом або після вбивства виводиться екран статистики та лунає звук виходу з гри. Екран після бою висновок статистики: отримані поранення у цьому бою; хто поранив останній.

Рисунок 4.1 – Приклад реалізації екрана у Форпост

Комплектація гвинтівки має включати:

– датчик ураження на радіо зв'язку;
– курок;
– динамік;
– кнопка примусового перезаряджання;
– лазерний приціл у корпусі на основі простої лазерної указки;
– ІЧ діод-випромінювач;
– маркер пострілу, як і був підсвічування;
– можливість підключення імітатора віддачі;
– LCD екран.

Датчик-приймач повинен мати індикацію підключення до гвинтівки - орієнтуватимемося по екрану гвинтівки. Живлення від одного елемента Li-Ion .

Зв'язок датчика з гвинтівкою на радіо. У кожній коробочці необхідні діоди яскраве підсвічування. За відсутності зв'язку гвинтівки з датчиком, ми повинні бачити на екрані. У разі втрати зв'язку датчика з гвинтівкою. Гвинтівка видає помилку на екрані та видає звук помилки та перестає стріляти.

Базова станція-пульт збирає статистику по РЧ-зв'язку передає на ноутбук який і зберігає у вигляді файлу диск.

Інформація зберігається за раундами в кінці документа, технічна частина та підсумкова статистика на початку файлу.

Управління пультом здійснюється за допомогою програмного інтерфейсу на комп'ютері, який підключений до базової станції послідовного порту.

*Меню та керування пультом повинне включати: «*Старт», «Стоп-раунд», «Стоп гра (кінець роботи з групою)», *«*Параметри гри» (Життя, Час раунду), Параметри зброї (номер або ID гвинтівки, пострілів у черзі, патронів в обоймі, магазинів із собою, сила пострілу, перезарядка – комбінована, ручна або автоматична, гучність 0-100 , віддача - вкл / викл , потужність ІЧ випромінювання.

Працюючи без пульта використовуються стандартні параметри: 3 життя, сила пострілу 1, напівавтоматична перезарядка, без кольору команди, дружній вогонь активний.

Звіт у гвинтівці під час раунду: мій номер (id), ім'я гравця,пострілів зроблено, поранень отримано, вбивств отримано, шкоди завдано (життя відібрано у противника), фраги (вбивств зроблено), номери стовбурів яких прийшли поранення, номер від якого прийшло вбивство.

Звіт надсилає гвинтівка на запит базової станції. Ім'я гравця задається наперед, постійне. На комплекті або на плакаті прописані номери всіх гравців та надані їм імена. Зі звітів формуємо підсумкову статистику на комп'ютері.

4.2 Проектування архітектури системи

У відомі на даний момент системах LASERTAG використовується архітектура бездротової мережі типу «зірка», особливістю якої є зв'язок роутера як з кожною гвинтівкою, так і з кожним датчиком [19]. При цьому прямий зв'язок між зброєю й датчиком відсутній. Більш того прив'язка гвинтівки до датчика існує лише на рівні ПО базовій станції. Факт влучення зачитується лише тоді, коли датчик взмозі передати сигнал на базову станцію як показано на рис. 4.2.

Незважаючи на заявлений час розгортання в кілька хвилин, організаційні моменти, пов'язані із закріпленням за гравцем датчика, залишаються - адже по суті ми маємо процедуру організації локальної мережі.

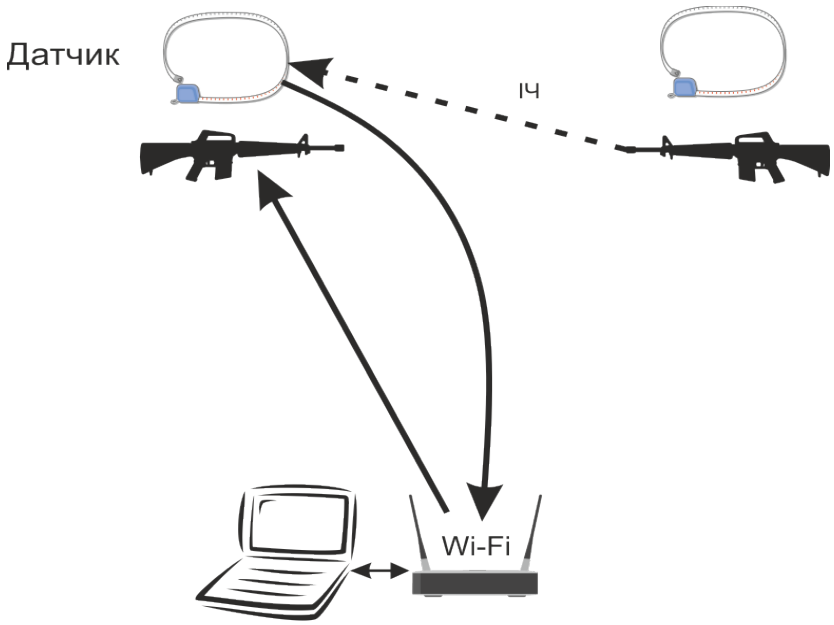

Рисунок 4.2 – Архітектура системи FORPOST

Інтерференція Wi-Fi з іншими пристроями може привести до різних негативних наслідків, що впливають на продуктивність і стабільність бездротової мережі. По-перше, вона може викликати зниження якості сигналу. Інтерференція від таких пристроїв, як мікрохвильові печі й бездротові телефони, значно погіршує якість сигналу Wi-Fi, особливо в діапазоні 2,4 ГГц, що призводить до втрати даних і зниженню швидкості передачі. По-друге, можуть виникнути проблеми з підключенням - пристрої, що працюють на тих же частотах (наприклад, Bluetooth), створюють перешкоди, що викликає нестабільне з'єднання та труднощі з підключенням до мережі Wi-Fi. По-третє, інтерференція може збільшувати затримку. Через цей час відгуку мережі може збільшуватися, що особливо помітно при використанні додатків, що вимагають високої швидкості передачі даних, таких як онлайн-гри. По-четверте, в умовах сильної інтерференції можливо повне відключення від мережі Wi-Fi, що може викликати значні незручності для користувачів. По-п'яте, знижується

продуктивність інших бездротових пристроїв. Наприклад, при активній передачі даних через Wi-Fi може погіршуватися якість зв'язку Bluetooth, таких як бездротові навушники або динаміки. Нарешті, для боротьби з інтерференцією може знадобитися установка додаткових маршрутизаторів або використання пристроїв з підтримкою більш високих частот (наприклад, 5 ГГц), що збільшує витрати [22-24]. Крім того неможна збільшувати потужність наголовних датчиків без шкоди здоров'ю. Крім того, перетнута місцевість (наприклад, будівлі) може бути причиною переривання зв'язку з роутером або суттєво зменшувати дистанцію впевненого прийому, що може призвести до руйнування мережі. Ці наслідки підкреслюють важливість обліку потенційних джерел інтерференції та перешкод при проектуванні й експлуатації систем LASERTAG в умовах імітації реального бою (накштальт, гра у штурм будівлі).

Щоб по-можливості уникнути перерахованих вище недоліків і домогтися автономії бійця (тобто спілки гвинтівка - датчик) без руйнування цілісності системи була розроблена деревоподібна архітектура, яка надана на рис. 4.3. Незважаючи на те, що на перший погляд ця система схожа, є ряд істотних моментів. По-перше, є жорсткий зв'язок між гвинтівкою та її датчиком (або датчиками), що формує автономний комплект бійця. По-друге, немає необхідності підтримувати безперервний зв'язок із базовою станцією під час бою - тільки на початку та наприкінці гри. Це означає, що логіка гри не страждатиме, якщо боєць спробує сховатися за бетонною плитою – його комплект також реагуватиме на влучення.

Проаналізувавши архітектуру системи, а також технічні та користувальницькі вимоги до неї, приходимо до висновку, що блок-схема гвинтівки повинна мати вигляд, показаний на рис. 4.4. Де CPU – це мікроконтролер STM32F405RG, NRF – РЧ модуль NRF24L01-SMA, ІЧ-приймач-TSOP572, ІЧ-діод – TSAL6100, LCD – Дисплей 2,2 дюйма на драйвері ILI9341. При цьому NRF та дисплей приєднано кожен через окремі канали SPI ІЧ-приймач приєднано до одного з входів USART з апаратною підтримкою кодування SIRC.

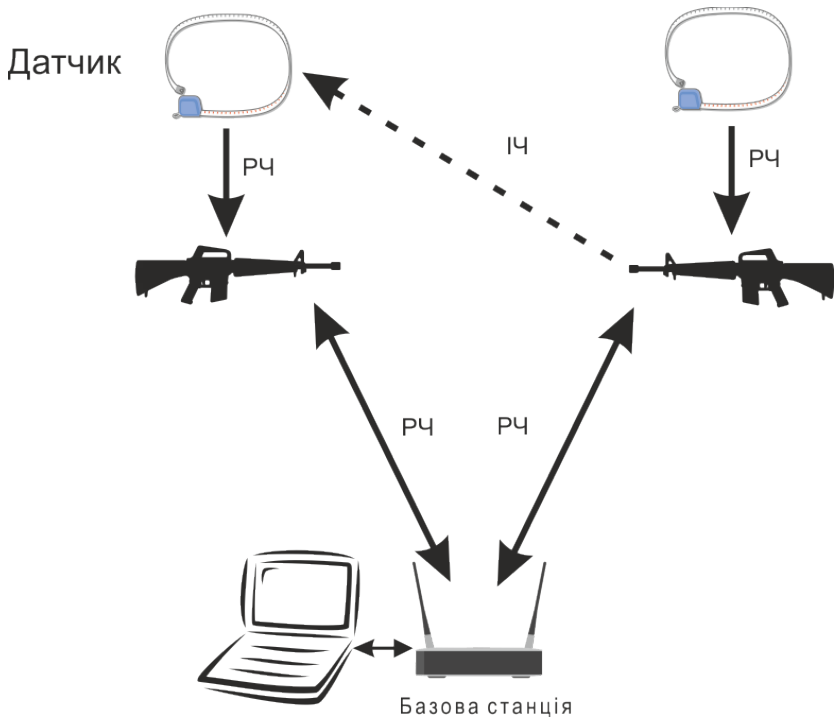

Рисунок 4.3 – Архітектура системи LASERTAG “ARENA”

РЧ модуль NRF24L01-SMA відрізняється від стандартного зовнішньої антеною та вбудованим підсилювачем, що дозволяє збільшити зону прийому до одного кілометра.

Блок-схема базової станції, який показаний на рис. 4.5, має схожу структуру, тому було вирішено розробляти уніфікований модуль на основі модуля гвинтівки. Тобто передбачити низку роз'ємів для підключення периферії, а також до RS-232 (наприклад, з допомогою драйвера- перехідника FS-232). Датчик влучення в цілях мініатюризації зібраний на іншому мікроконтролері - STM32F030F4P6. До нього не пред'являються вимог щодо продуктивності, тому що він служить лише для ретрансляції ІЧ-сигналу, проте він має досить непогані характеристики: 48 МГц тактової частоти, 16 Кбайт Flash, 4 Кбайта SRAM. В даному випадку ІЧ-приймач – це до чотирьох TSOP572, які приєднані індивідуально. Для індикації влучення передбачений над’яскравий синій світлодіод, під'єднаний через драйвер на КМОП транзисторі, аналогічно під'єднаний вібромотор для індикації влучення. Якщо пов'язка одягнена на голову – гравець відчуватиме кожне влучення в нього.

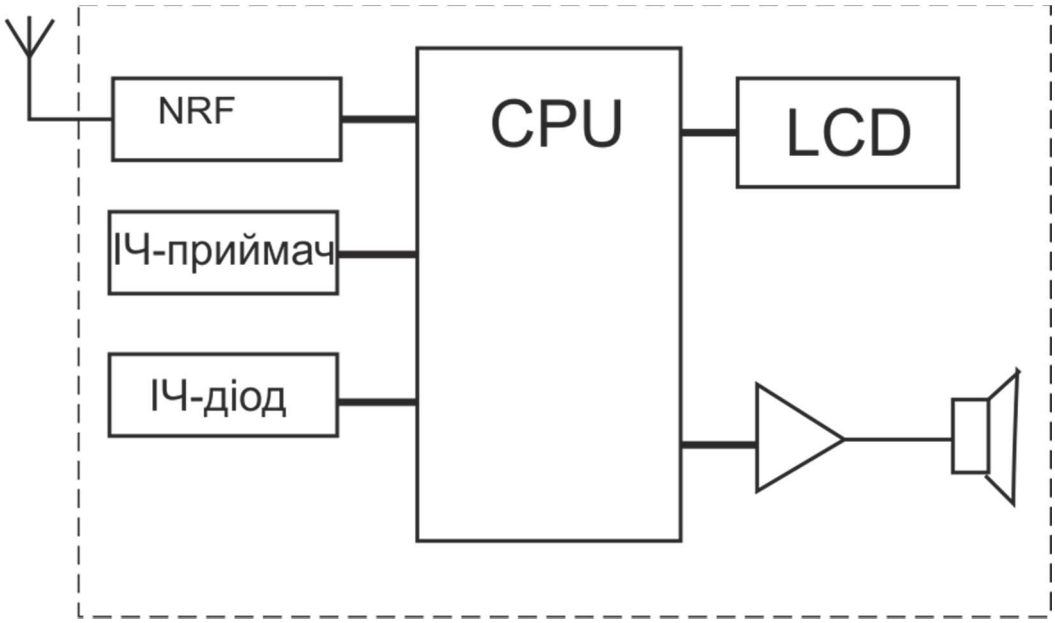

Рисунок 4.4 – Блок-схема гвинтівки

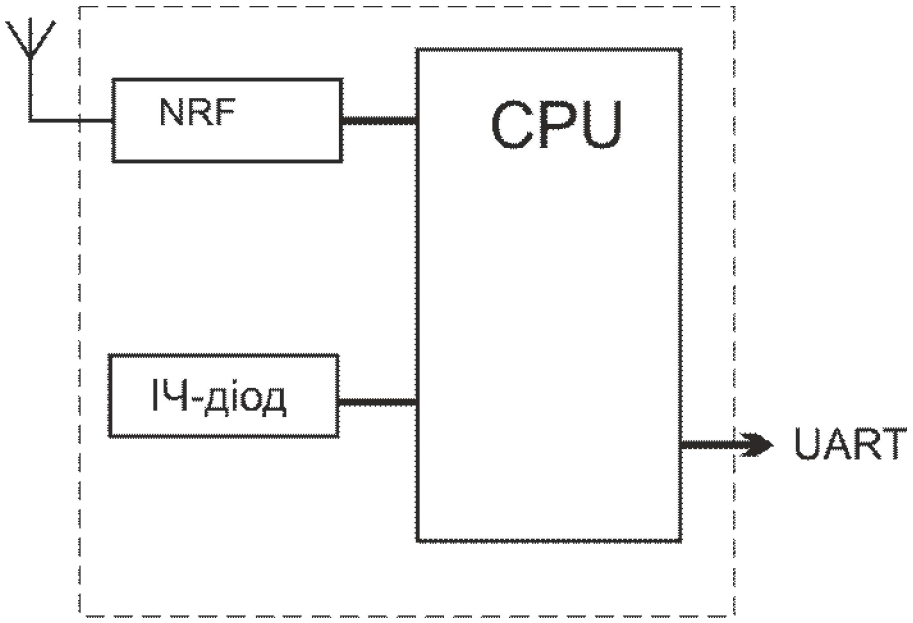

Рисунок 4.5 – Блок-схема базової станції

Згідно з наведеними вище блок схемами розроблені схеми електричні принципові універсального керуючого блока та датчику влучення, які наведені в додатку А.

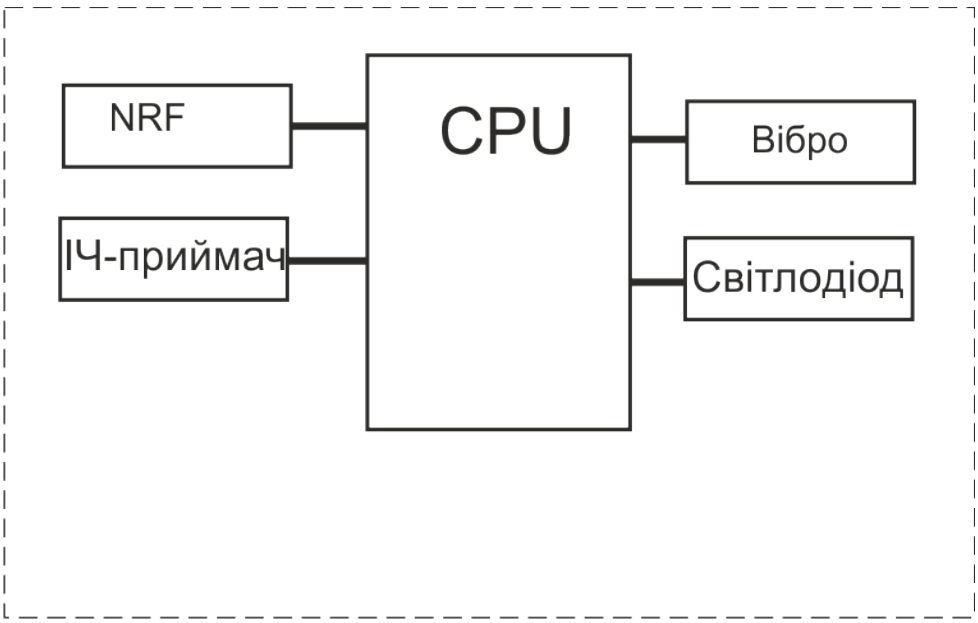

Рисунок 4.6 – Блок-схема датчику влучення

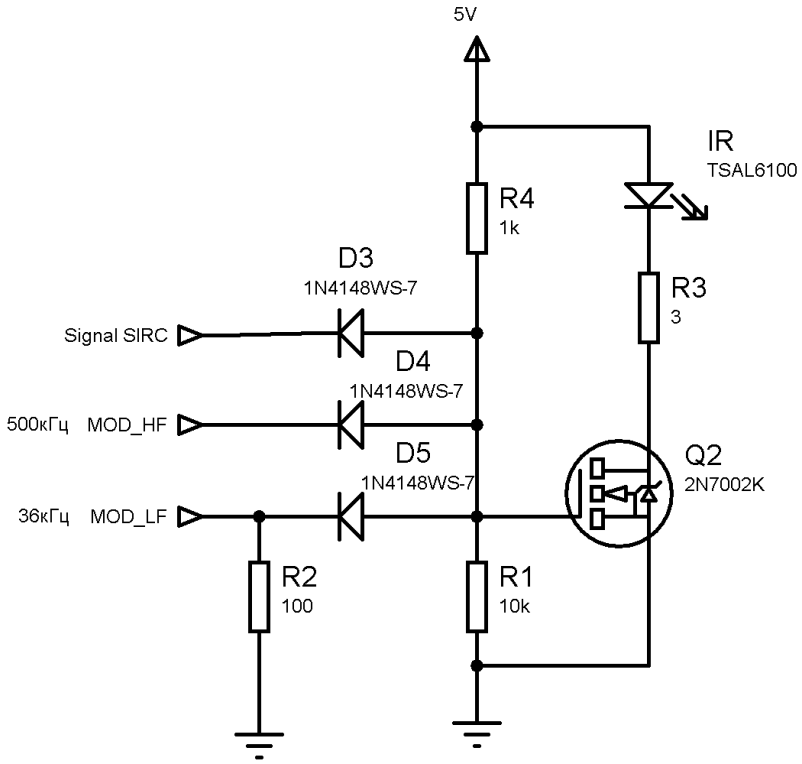

Рисунок 4.7 – Схема підключення ІЧ-світлодіоду

Схема підключення ІЧ-світлодіоду надана на рис. 4.7. Вхідний сигнал SIRC модулюється несучою 36кГц, та надмодулюється ШІМ 500кГц. Це дозволяє регулювати потужність світлодіода.

5 РОЗРОБКА ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

5.1 Проектування архітектури програмного забезпечення

Якщо скористатися методом декомпозиції, то програмне забезпечення моделі гвинтівки можна представити у вигляді трьох шарів: найнижчий – рівень переривань, регістрів пристроїв та STM 32 StdLib , наступний – бібліотеки драйверів периферії, які можуть взаємодіяти шляхом надсилання повідомлень та останній – об'єктна модель програмного середовища TaskManager, яка підтримує обробку повідомлень та запуск підзадач: TServicemodeApp , TDefaultmodeApp , TFightmodeApp (див. рис. 5.1). Кожна з цих підзадач відповідає режиму, в якому знаходиться гвинтівка – відповідно сервісний, режим очікування та режим бою. Завдання TaskManager - а у забезпеченні послідовного запуску режимів та організації середовища виконання

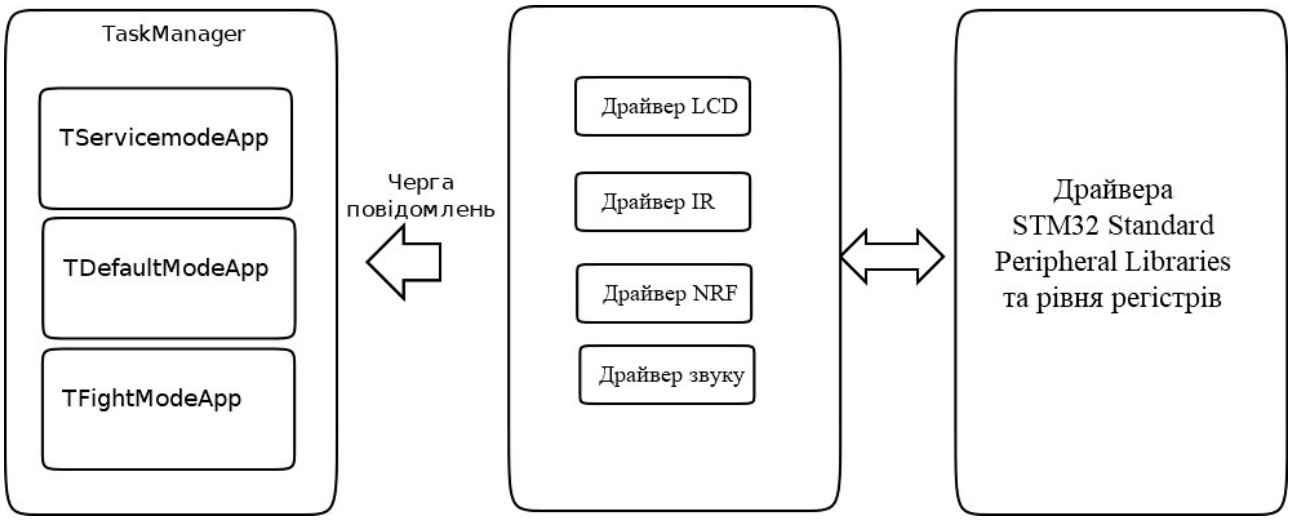

Рисунок 5.1 – Декомпозиція ПЗ гвинтівки

Об'єктна модель надана на рис. 5.2. Менеджер завдань представлений абстрактним класом TTaskShell (в C ++ немає абстрактних класів у розумінні цього слова, тому цей термін вживається для класів, з яких не будуть вироблятися екземпляри). TTaskShell є предком для TRifle , який реалізує поверх нього інформаційну модель рушниці (конфігурацію).

TBaseAppClass також є абстракцією підзавдання, яка в нескінченному циклі функції Run( ) отримує та обробляє повідомлення (див. лістинг 5.1).

Лістинг 5.1 – Ключовий метод Run класу TBaseAppClass

```
void TBaseAppClass ::Run()
{
     TMessage Message;
     Terminated = false;
     while( !Terminated)
     {
          if ( GetMessage (Message))
          {
               OnMessage ( Message);
               Dispatch ( Message);
          }
          else OnCPUIdle();
     }
}
```

Функція bool GetMessage (TMessage & Message) звертається до циклічної черги повідомлень і якщо повідомлення отримане – витягує його. Метод Dispatch( ) – центральний у обробці повідомлень – його мета перенаправити повідомлення його обробнику (тобто згенерувати програмну подію). Спадкоємці TBaseAppClass повинні перекривати цей метод для обробки своїх подій. Наприклад, TBaseAppClass генерує подію virtual void OnMessage(), яке можна перекрити при наслідуванні і перехопити.

Клас TTask – наступний рівень абстракції – цей клас більш орієнтований реалізацію, оскільки містить у собі посилання середу виконання – поле TRifle *rifle . Таким чином, його спадкоємці можуть отримувати доступ до методів екземпляра rifle (середовища виконання). Це дозволяє розділяти загальну пам'ять та процедури між режимами (процесами). Крім того, цей клас містить методи ініціалізації обладнання – InitDrivers(), InitDisplay(). Так як ці методи віртуальні, вони можуть бути перевизначені спадкоємцями для своїх цілей. Наприклад, TDefaultmodeApp, TFightmodeApp перевизначають метод InitDisplay( ) для організації своїх елементів індикації на екрані .

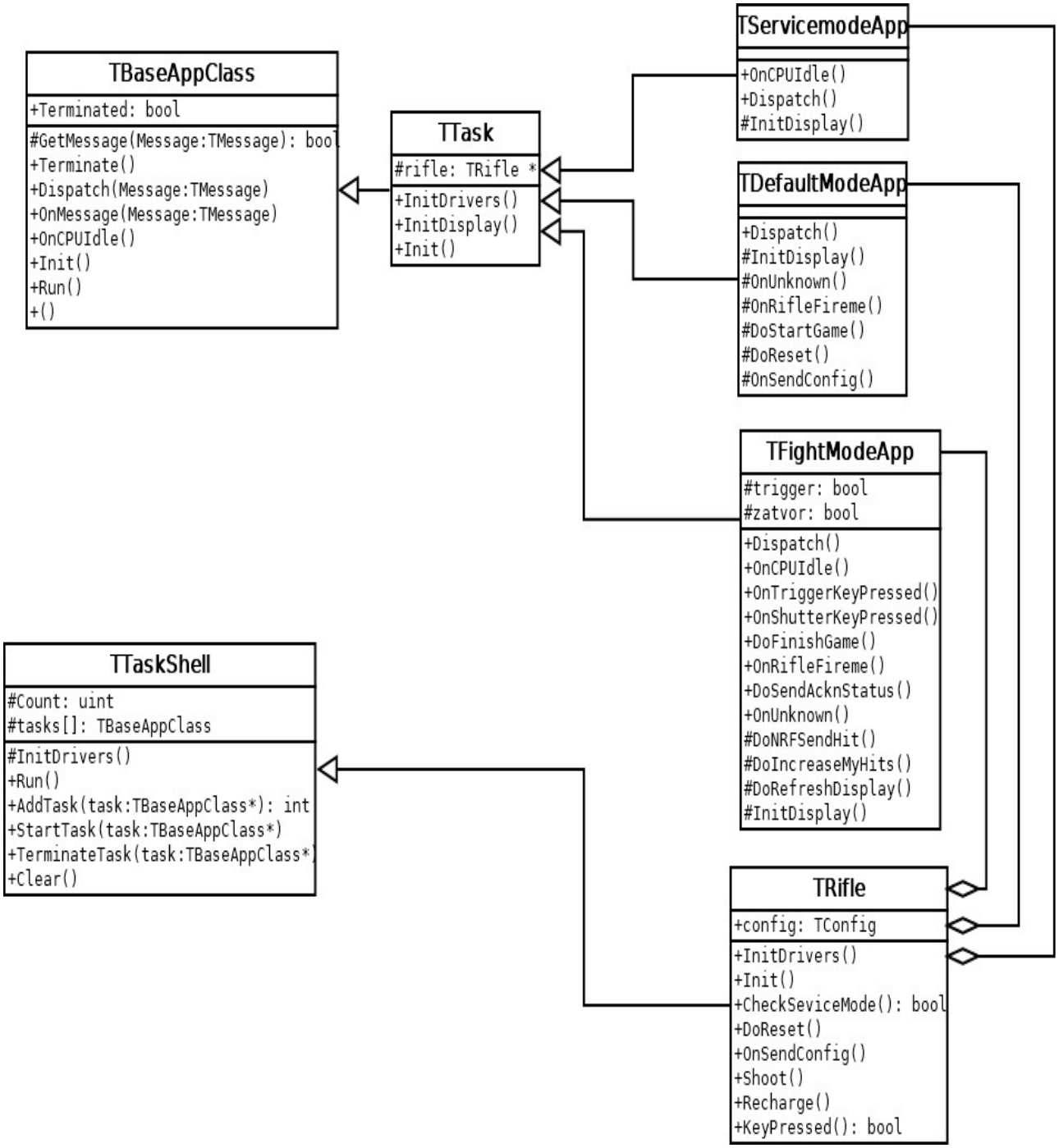

Рисунок 5.2 – Діаграма класів ПЗ гвинтівки

Перерахуємо деякі з методів класу TFightModeApp, які реалізують реакції на повідомлення (події):

– OnTriggerKeyPressed() – реакція на гачок;

– OnShutterKeyPressed() реакція на кнопку перезарядки;

– DoFinishGame() отримана команда от пульта на кінець гри;

– OnRifleFireme() – в мене влучили – звук, індикація, втрата життя;

– DoNRFSendHit() перешлемо по РЧ, що у нас влучілі;

– DoSendAcknStatus() // nrf запитує команду у acknoledment-е;

– DoIncreaseMyHits() наростимо наш лічильник влучень.

Перерахуємо деякі з методів класу TRifle. Ці загальні методи, що використовуватимуся подзадачами:

– bool CheckSeviceMode() – перевірка на можливість вийти у сервісний режим;

– void DoReset() – скидання стану;

– void OnSendConfig() – надіслані параметри з пульта, записуємо у пам’ять;

– void Shoot() – макрос пострілу;

– void Recharge() – макрос перезарядки;

– bool KeyPressed(int keycode) – перевірка натискання на кнопку.

Оскільки архітектура ПЗ у зброї є подієва, то методи, які розглянути вище, надають уявлення про алгоритм роботи гвінтівки.

Зараз розглянемо як працює датчик. Алгоритм роботи датчика влучення наступний.

Крок 1. Ініціалізація. Після включення відбувається ініціалізація портів введення-виводу.

Крок 2. Сервісний режим. Триває п'ять секунд. Очікується ІЧ-пакет для прив'язки до гвинтівки N.

Крок 3.Цикл передачі:

– очікується закінчення прийому пакета даних від ІЧ;

– якщо пакет прийнято – передаємо гвинтівці по РЧ-каналу номер N, пакети не перевіряємо та не дешифруємо;

– якщо протягом 100мс не прийнято жодного пакета – провокуємо гвинтівку на acknowledgment передачею порожнього пакета. сприймаємо acknowledgment як команду управлінням периферією – мотором та світлодіодом.

5.2 Реалізація кодування Ріда-Соломона

Існує спосіб передачі даних, при якому можливо відновити втрачені дані після прийому. Це, накштальт, коди Ріда-Соломона. Припустимо, у вас є повідомлення: "DON'T PANIC". Якщо ви хочете додати до нього кілька надлишкових байтів, скажімо, 6 частин: "RRRRDON'T PANIC" (кожен r - це байт, обчислений алгоритмом), перенесіть його на який -небудь носій з перешкодами і шість байт виявилися помилкою, можна повністю відновити вихідне повідомлення. зайві символи, що коректують.

Як правило, можна додати до повідомлення будь-яку кількість зайвих символів за необхідності. Кількість зайвих символів збігається з кількістю виправлених помилок (тільки у разі, якщо відомий номер місця виявлення помилки). Як правило, помилки у відомих місцях називаються стирання.

Крім того, у коді Ріда-Соломона можна виправляти помилки з невідомим розташуванням, але для виправлення кожної помилки потрібно 2 зайві символи. "RRRRDO___PANIC", який приймається як " RRRRRDON't PANIC", може бути легко виправлений без додаткової інформації. Неправильно отриманий байт, місцезнаходження якого невідоме, у майбутньому називатиметься "помилкою" і.

Можна поєднати виправлення помилки та помилки. Важливо, що для виправлення помилки потрібно якимось чином знайти номер байта помилки. Важливий факт, що "помилка" та "друкарська помилка" також можуть бути виправлено за допомогою алгоритму додаткових байт.

Очевидно, якщо кількість відправлених і отриманих байт відрізняється, то код Ріда-Соломона тут фактично безсилий Іншими словами, якщо місцезнаходження втраченого символу невідоме, ніяка операція не може бути завершена.

5.2.1 Кодування повідомлення

Необхідно як висловлювати повідомлення у вигляді багаточленів так і виконувати над ними операції додавання, множення та поділу див. формулу (5.1):

$$(a^1 - x) \cdot (a^2 - x) \cdot \cdots \cdot (a^M - x), \tag{5.1}$$

де a – це вихідний елемент поля (зазвичай вибирають 2), а M – це кількість надлишкових символів.

Тобто, перш ніж створювати код Ріда-Соломона з повідомлення, ви повинні визначити кількість надлишкових символів, яка, на вашу думку, є достатньою. Один і той же генератор поліномів може бути використаний для будь-якого повідомлення, і будь-яке повідомлення в цьому випадку буде закодовано однаковою кількістю надлишкових символів характеристикою два, ми можемо сміливо писати знак плюс замість знаку мінус:

$$(2^1 + x) \cdot (2^2 + x) \cdot (2^3 + x) \cdot (2^4 + x) = 116 + 231x + 216x^2 + 30x^3 + x^4 \tag{5.2}$$

Наведений вище вираз є генераторним поліномом, який повинен кодувати повідомлення будь-якої довжини з додаванням 4 надлишкових символів і може призвести до двох "помилок" або чотирьох "друкарських помилок".

Насправді, при написанні програми достатньо знати коефіцієнти многочлена, а ступінь можна дізнатися з цих коефіцієнтів. Таким чином, багаточлен генератора, отриманий у наведеному вище прикладі, може бути записаний як: {116, 231, 216, 30, 1}. Крім того, щоб зробити його більш компактним, ви можете опустити дужки та коми і записати все у поданні base16: 74 E7 D8 1E 01. Отже, беручи до уваги вищезгаданий протокол, достатньо записати цей байт, щоб висловити повідомлення "без паніки" у поліноміальній формі: 44 4F 4E 27 54 20 50 41 4E 49 43.

Щоб створити код Ріда-Соломона з 4 надмірними символами, перемістіть багаточлен праворуч на 4 позиції (що еквівалентно його множенню): 00 00 00 00 44 4F 4E 27 54 20 50 41 4E 49 43. Розділимо отриманий багаточлен на багаточлен генератора (74 E7 D8 1E 01), візьміть частину від поділу (DB 22 58 5C) і напишемо багаточлен від поділу замість нуля,що еквівалентно операції складання: DB 22 58 5C 44 4F 4E 27 54 20 50 41 4E 49 43.

Цей рядок стане кодом Ріда-Соломона для повідомлення "DON'T PANIC" із чотирма зайвими символами. Також під час створення коду можна ділити на полином-генератор, отримуючи залишок, а множити нього. Це трохи інший різновид коду Ріда-Соломона.

5.2.2 Розкодування повідомлення

Розкодувати просто – прибираємо надлишкові символи і залишається вихідне повідомлення. Насамперед слід зазначити, що з перевірці на наявність помилок потрібно знати кількість надлишкових символів. А по-друге – треба навчитися вважати значення полінома за певного x . Про кількість надлишкових символів нам повинен заздалегідь повідомити той, хто кодував повідомлення, а щоб обчислити значення полінома потрібно написати ще одну функцію для роботи з поліномами. Отже, перш ніж виправляти "помилки" або "друкарські помилки" потрібно дізнатися чи є вони. Потрібно обчислити поліном прийнятого повідомлення з надмірними символами при рівних ступенях примітивного члена. Це ж числа, які ми використовували при складанні полінома-генератора. Якщо помилок немає, то всі обчислені значення дорівнюватимуть нулю. Закодоване раніше повідомлення «DON'T PANIC» із чотирма надмірними символами, у вигляді полінома у шістнадцятковому поданні: DB 22 58 5C 44 4F 4E 27 54 20 50 41 4E 49 43, якщо обчислити цей поліном при x рівному 2, 4, 8, 16, вийдуть значення: 0, 0, 0, 0, адже тут повідомлення точно в такому ж вигляді, в якому воно і було закодовано. Якщо змінити хоча б один байт, наприклад, останній символ зробимо правильнішим: 42 замість 43: DB 22 58 5C 44 4F 4E 27 54 20 50

41 4E 49 42, то результат такого ж обчислення дорівнюватиме 13, 18, B5, 5D. Ці значення називають синдромами. Тоді це буде поліном синдромів.

Отже, щоб дізнатися, чи є помилки в прийнятому повідомленні, потрібно вважати поліном синдромів. Якщо він складається з одних нулів (також можна говорити, що він дорівнює нулю), то помилок немає. Може статися так, що повідомлення з помилками матиме синдром рівним нулю. Це трапиться в тому випадку, коли поліном амплітуд помилок (про нього буде нижче) кратний поліном-генератору. Отже, перевірку помилок по поліному синдромів коду Ріда-Соломона не можна вважати 100% гарантією відсутності помилок. Можна навіть вважати ймовірність такого випадку.

Допустимо ми кодуємо повідомлення із 4 символів чотирма ж надлишковими символами, тобто передаємо 8 байт. Також візьмемо для прикладу можливість помилки при передачі одного символу в 10%. Тобто в середньому на кожні 10 символів припадає один, який передався як випадкове число від 00 до FF. Це, звичайно ж, зовсім синтетична ситуація, яка навряд чи буде в реальності, але тут можна точно обчислити ймовірності.

Поліноми, кратні поліном-генератору виходять множенням генератора інші поліноми. П'ятизначний кратний поліном – виходить множенням на константу від 1 до 255. Шестизначний - множенням на бином першого ступеня. Ті ж міркування для 7 і 8-значних поліномів, кратних генератору. Потім треба знайти ймовірності випадання 5, 6, 7 і 8 помилок поспіль, і для кожної з них обчислити ймовірність, що така випадкова послідовність помилок виявиться кратною поліном-генератору. Скласти їх, і тоді ми отримаємо ймовірність того, що при передачі 4 байт з 4 надмірними символами, при ймовірності помилки при передачі одного символу 10% вийде помилкова передача, що не виявляється кодом Ріда-Соломона. Проведемо розрахунок, який дає зрозуміти про порядки чисел.

Кількість комбінацій по k з n:

$$C(n,k) = n!/k! \cdot (n-k)! \tag{5.3}$$

Формула Бернуллі:

$$PB(n,k,p) = C(n,k) \cdot p^k \cdot (1-p)^{n-k}. \quad (5.4)$$

Кількість можливих k результатів підряд в послідовності довжиною n:

$$S(n,k) = n - k + 1. \quad (5.5)$$

Імовірність P_k появи результату k раз підряд у n випробувань при ймовірності події p:

$$P_k = PB(n,k,p) \cdot S(n,k)/C(n,k)\ . \quad (5.6)$$

Кількість N-символьних кратних 5-символьному генератору поліномів рівно:

$$Q_5(N) = 255^{N-5+1}. \quad (5.7)$$

Імовірність появи послідовності хибно-безпомилкової при N випадкових символах підряд для 5-символьного генератора:

$$P_f(N) = Q_5(N)/256^N. \quad (5.8)$$

Імовірність появи хибно-негативного тесту на помилку при передачі 4 байт із 4 надлишковими символами, і ймовірності випадкової помилки в 1 байті 10%:

$$Err = \sum_{k=5}^{8} \left(P(8,k,0.1) \cdot P_f(k) \right) = 7.367 \cdot 10^{-15}. \quad (5.9)$$

Скільки терабайт потрібно передавати, щоб наткнутися на такий випадок (5.10).

$$4 \cdot Err^{-1}/1024^4 \approx 494 \text{ Тб}. \quad (5.10)$$

Тобто на кожні ~500 Тб при такій передачі виявиться один блок з четирьох помилкових символів, які алгоритм вважатиме коректними.

Якщо синдром не дорівнює нулю, необхідно виправляти помилки. Розглянемо випадок з "друкарськими помилками", коли ми точно знаємо номери позицій некоректно прийнятих байт.

Цього достатньо, щоб відновити повідомлення у вихідний стан. Для продовження необхідно розучити ще одну операцію з поліномами в полях Галуа - взяття формальної похідної полінома. Формальна похідна полінома у полі Галуа схожа на звичайну похідну. Формальною вона називається тому, що в полях на кшталт GF[256] немає дробових чисел, і відповідно не можна визначити похідну, як відношення нескінченно малих величин. Обчислюється схоже звичайну похідну, але з особливостями. Якщо при звичайному диференціювання

$$(ax^n)' = a \cdot n \cdot x^{n-1}, \quad (5.11)$$

то для формальної похідної в полі Галуа з основою два формула для диференціювання члена така:

$$(ax^n)' = a(n \bmod 2) \cdot x^{n-1}. \quad (5.12)$$

Це означає, що остаточно просто переписати поліном, починаючи з першого ступеня (нульова викидається) і у того, хто залишився обнулитичлени з непарними ступенями. Наприклад, потрібно знайти похідну в шістнадцятковому вигляді, що виглядає так: (01 2D A5 C6 8C DF) ' = 2D 00 C6 00 DF.

З прикладу у шістнадцятковому вигляді найпростіше скласти алгоритм знаходження формальної похідної.

Крім того, потрібно ще два поліноми: поліном-локатор і поліном помилок.

Поліном-локатор – це поліном, корінням якого є числа зворотні примітивному члену в позиції помилки:

$$(1 + x \cdot a^{E_1}) \cdot (1 + x \cdot a^{E_2}) \cdot \cdots \cdot (1 + x \cdot a^{E_N}), \quad (5.13)$$

де a – це примітивний член, E_1 , E_2 , та тощо – це позиції помилок.

Поліном-визначник має наступне: з нього можна отримати положення помилки і навпаки. По суті, це два різні записи того самого – положення помилки. У різних статтях поліноми помилок називаються по-різному, але не так складно. Це добуток полінома синдрому та полінома локатора, і вищі ступені відкидаються.

Розрахуємо амплітуду помилки. Це значення, які потрібно додати до спотвореного символу повідомлення, щоб отримати неспотворений символ. Для цього скористаємося алгоритмом Форні [26] . Наведемо фрагмент коду, щоб описати це (див. лістинг 5.2).

Лістинг 5.2 – Функция, що розраховує амплітуду помилки

```
public static GF_Poly FindMagnitudesFromErrPos (
   GF_Poly Syndromes ,
   GF_Poly ErrPos ,
   uint NumOfErCorrSymbs )
{    GF_Poly Locator = CalcLocatorPoly ( ErrPos );
     GF_Poly Product = Syndromes * Locator ;
     GF_Poly ErrPoly = Product.DiscardHiDeg ( NumOfErCorrSymbs );
     GF_Poly LocatorDer = Locator.FormalDerivative ();
     GF_Poly Magnitudes = new GF_Poly ( ErrPos.GetMaxCoef ());
     for ( uint i = 0; i < ErrPos.Len ; i++)
{         GF_Byte Xi = 1 / GF_Byte.Pow_a ( ErrPos [i]);
          GF_Byte W = ErrPoly.Eval ( Xi );
          GF_Byte L = LocatorDer.Eval ( Xi );
          GF_Byte Magnitude = W/L;
          Magnitudes [ ErrPos [i]] = Magnitude ;
     }
     return Magnitudes ;
}
```

Функція приймає на вході: поліном синдромів (Syndromes),поліном позиції помилок (ErrPos), кількість надлишкових символів (NumOfErCorrSymbs).

Клас Gf_Poly – це багаточлен у полях Галуа. Він також перевизначає арифметичні операції таким чином, щоб вони виконувались відповідно до правил арифметики з поліномами Галуа.

Якщо ви вводите у функцію відповідні параметри – на виході вона дасть поліном амплітуд помилок. Якщо його додати до спотвореного повідомлення, відповідно до правил складання багаточленів, – результатом буде вихідне повідомлення.

Бажано мати можливість ідентифікувати пошкоджені байти без додаткової інформації. Для цього необхідно вміти точно ідентифікувати байти, в яких біт був пошкоджений під час передачі.

Поліном-визначник може бути складений на основі відомих місць помилок, а також може бути розрахований на основі полінома синдрому і кількості надлишкових символів. Існує кілька алгоритмів, які дозволяють це зробити. Це код реалізує алгоритм Берлекампа- Мессі [24] (лістинг 5.3).

Лістинг 5.3 – Алгоритм Білкампа Мессі

```
  bool FindErrorLocator ( const Poly * synd , Poly * erase_loc =
NULL, size_t erase_count = 0) {
        Poly * error_loc = & polynoms [ ID_ERRORS_LOC];
        Poly * err_loc = & polynoms [ ID_TPOLY1];
        Poly * old_loc = & polynoms [ ID_TPOLY2];
        Poly * temp = & polynoms [ ID_TPOLY3];
        Poly * temp2 = & polynoms [ ID_TPOLY4];

        if ( erase_loc ! = NULL) {
            err_loc -> Copy ( erase_loc );
            old_loc -> Copy ( erase_loc );
} else {
            err_loc -> length = 1;
            old_loc -> length = 1;
            err_loc -> at ( 0) = 1;
            old_loc -> at ( 0) = 1;
}

uint8_t synd_shift = 0;
        if ( synd -> length > ecc_length ) {
            synd_shift = synd -> length - ecc_length ;}

        uint8_t K = 0;
        uint8_t delta = 0;
```

```
        uint8_t index ;

        for ( uint8_t i = 0; i < ecc_length - erase_count ; i++){
            if ( erase_loc ! = NULL)
K = erase_count + i + synd_shift ;
            else
K = i + synd_shift ;

            delta = synd -> at (K);
            for ( uint8_t j = 1; j < err_loc -> length ; j++) {
                index = err_loc -> length - j - 1;
              delta ^ = gf :: mul ( err_loc -> at ( index ),
           synd-> at (Kj));
}

            old_loc -> Append ( 0);

            if ( delta ! = 0) {
                if ( old_loc -> length > err_loc -> length ) {
                    gf :: poly_ scale ( old_loc , temp , delta );
                    gf :: poly_ scale ( err_loc , old_loc , gf ::
inverse ( delta ));
                    err_loc -> Copy ( temp );
}
                gf :: poly_ scale ( old_loc , temp , delta );
                gf :: poly_ add ( err_loc , temp , temp2);
                err_loc -> Copy (temp2);
}
}

        uint32_t shift = 0;
        while ( err_loc -> length && err_loc -> at ( shift ) == 0)
shift ++;

        uint32_t errs = err_loc -> length - shift - 1;
        if ( (( errs - erase_count ) * 2 + erase_count )> ecc_length
) {            return false ; }
        memcpy ( error_loc -> ptr (), err_loc -> ptr () + shift , (
err_loc -> length - shift ) * sizeof (uint8_t));
        error_loc -> length = ( err_loc -> length - shift );
        return true;
}
bool FindErrors ( const Poly * error_loc , size_t msg_in_size ) {
        Poly * err = & polynoms [ID_ERRORS];

uint8_t errs = error_loc -> length - 1;
        err -> length = 0;

        for ( uint8_t i = 0; i < msg_in_size ; i++) {
            if ( gf :: poly_eval ( error_loc , gf :: pow (2, i)) ==
0) {
                err -> Append ( msg_in_size - 1 - i);
}
```

```
}

        if ( err -> length ! = errs )
            return false ;
        return true ;
}
```

Наведений вище алгоритм підраховує локатор. Якщо кількість "помилок" перевищує кількість надлишкових символів, поділена на 2, алгоритм працюватиме некоректно. Поліном-визначник містить інформацію про місцезнаходження помилки. Тому що це "багаточлен, корінь якого є зворотною величиною від примітивного члена по відношенню до ступеня розташування помилки". Виходячи з цього, очевидно, що положення помилки дорівнює логарифму числа підстав примітивного члена, зворотному числу коренів многочлена:

$$E = \log_a(1/R), \tag{5.13}$$

де E – це положення помилки, a – примітивний член (як правило, 2), а R – корінь багаточлена.

Знайдемо коріння в полі Галуа. В GF[256] може бути всього 256 чисел. помилки. Тепер ці значення поряд з синдромом і числом 4 вводяться в алгоритм Форні , який дозволяє виправити помилку таким же чином, як і виправлення помилки.

Таблиця 5.1 – Формат даних пострілу

Команда	N зброї	Arg1	Arg2,	Код Соломона-Ріда
8 біт	8 біт	8 біт	8 біт	32 біта

Таким чином, формат даних, які відправляє зброя прі пострілі буде такім, що наведеній у табл. 5.1, де Команда – це тип посилання – наприклад, постріл; N– номер зброї, Arg1 – потужність пострілу, Arg2 – необов’язковий параметр.

5.3 Проектування інтерфейсу користувача

При розробці пристроїв на мікроконтролерах відображення графічної інформації відіграє ключову роль. У цьому контексті широко використовуються різні типи дисплеїв, такі як РК-дисплеї, TFT та OLED, які мають діагоналі від 1 до 6 дюймів. Незважаючи на компактні розміри, ці пристрої надають значні можливості для візуалізації як графічної, так і текстової інформації. Наприклад, 2,2-дюймовий TFT-дисплей з драйвером ILI9341 має роздільну здатність 240x320 пікселів і підтримує відображення 16-бітного кольору. Для роботи з такими дисплеями розроблені спеціальні низькорівневі бібліотеки, які дозволяють налаштовувати кольори пікселів та малювати графічні примітиви.

У цій системі відображення використовується об'єктно-орієнтований підхід мовою C++, що дозволяє ефективно управляти компонентами екрана (див. рис. 5.3). Базовим класом у цій архітектурі є TScreenComponent , який містить основні атрибути, такі як координати, розміри та видимість компонента. Він включає два ключові методи: virtual Draw( ), який відповідає за малювання компонента, та virtual Clear (), який очищає область компонента. Ці методи можуть бути перевизначені у спадкоємцях для реалізації специфічної поведінки.

Класи-спадкоємці TText , TPicture , та TProgressBar являють собою різні елементи інтерфейсу: текстові рядки, зображення та індикатори виконання відповідно. Це дозволяє створювати гнучку систему для відображення інформації на екрані мікроконтролера.

Наступним важливим елементом є абстрактний клас TCompositeScreenComponent , який може містити кілька інших компонентів. Це досягається за допомогою масиву Components [MAX_SCREEN_COMPONENTS] та методу Add(TScreenComponent*). Конструктор класу TScreenComponent може приймати покажчик на батьківський компонент, що дозволяє додавати екземпляри в колекцію TCompositeScreenComponent. Це забезпечує можливість проектування складних інтерфейсів на етапі розробки.

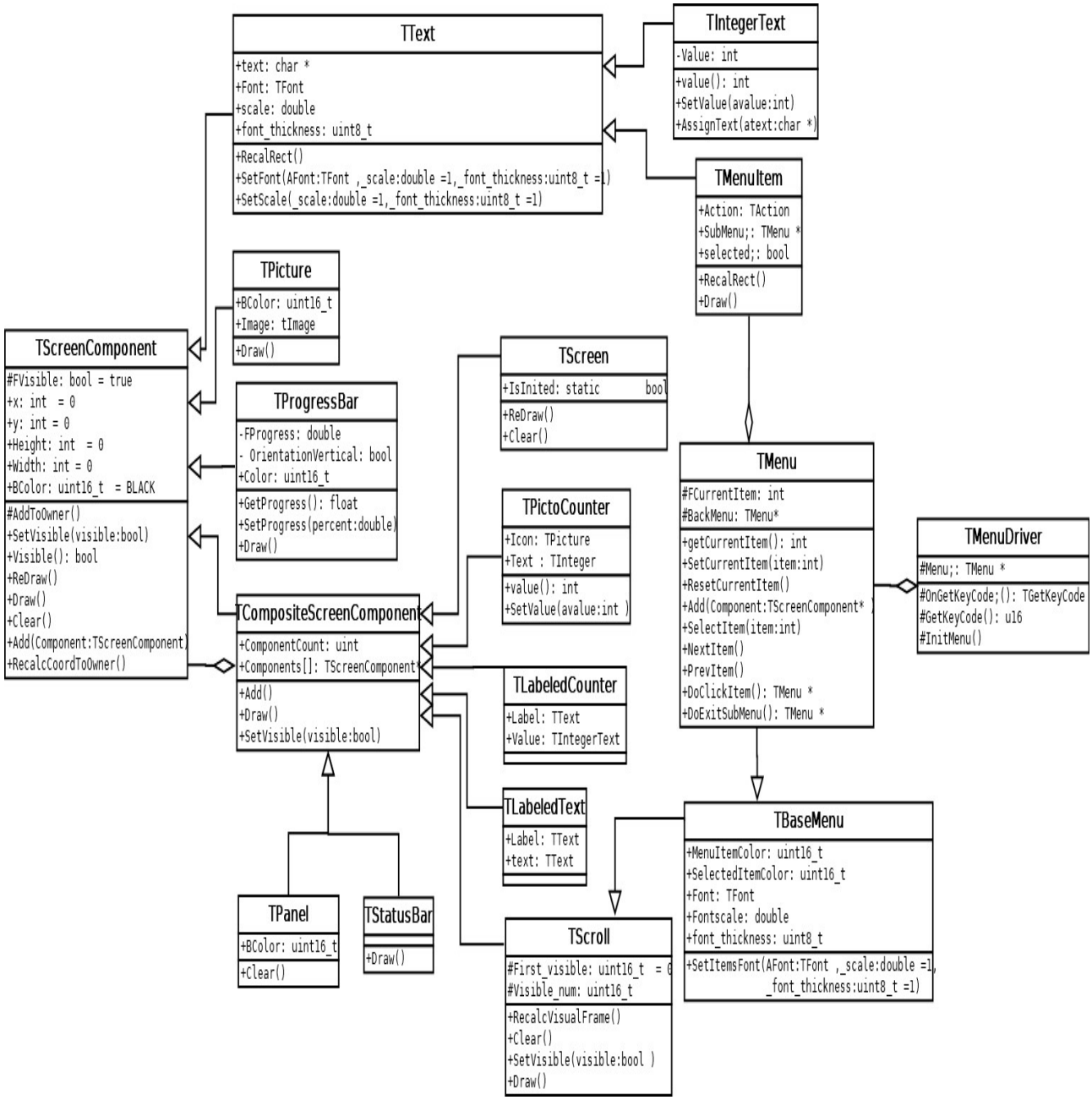

Рисунок 5.3 – Діаграма класів графічного інтерфейсу

Клас TPanel був введений для керування частинами екрану, де можуть бути інші компоненти. Він дозволяє динамічно змінювати вміст екрана, приховуючи одну панель та показуючи іншу. Наприклад, інтерфейс може включати рядок стану і дві панелі - Panel1 і Panel2, які є спадкоємцями класу TPanel. Це дозволяє легко керувати видимістю панелей за допомогою методу setVisible.

Об'єктно-орієнтований підхід до проектування інтерфейсів для мікропроцесорних пристроїв забезпечує високий рівень модульності та

повторного використання коду. Це особливо корисно у контексті розробки вбудованих систем, де ресурси обмежені, а вимоги до продуктивності високі. Використання C++ дозволяє розробникам створювати ефективні та легко підтримувані програми, які можуть адаптуватися до різних завдань.

Екран у режимі очікування має вигляд як на рис. 5.5. Екран у режимі бою виглядає надан на рис. 5.6.

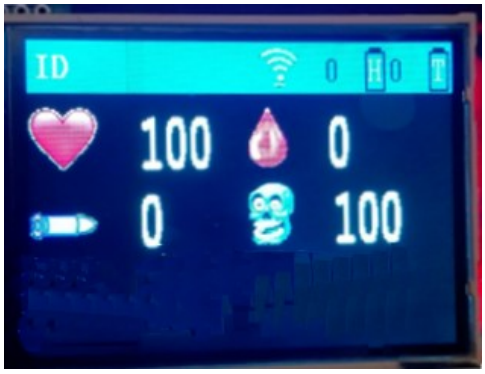

Рисунок 5.5 – Екран у режимі очікування

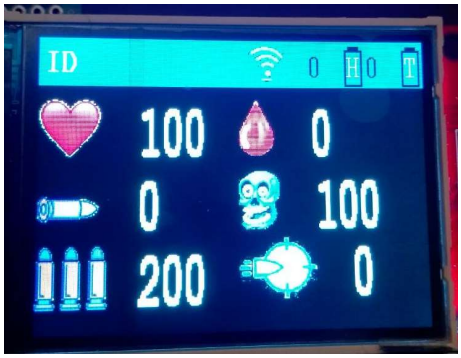

Рисунок 5.5 - Екран у режимі бою

Лістинг 5.3 – Створення екрана для режиму бою

```
class TPanel1: public TPanel
{
public:
      TPictoCounter Health,Hurt;
 TPictoCounter PatronCount, LastHurter;

     TPicture Packs;TPicture Packs1;TPicture Packs2; TIntegerText
PatronPackCount;
     TPictoCounter HitsCounter;
     TPanel1(int ax,int ay, uint16_t bcolor,TScreenComponent
*AOwner):
```

```
          TPanel(    ax,    ay,AOwner->Height-ax,AOwner->Width-ay,
AOwner),
               Health(0,0,ImageCardial,100,30,this,-5),
Hurt(160,0,ImageBloodDrop,0,30,this,-5),

     PatronCount(0,120,ImageHorPatron,0,30,this,-5,20),
     LastHurter(160,60,ImageBlueSkull,100,30,this,-5),
                              Packs(0,        60,ImagePatron,this),
Packs1(20, 60,ImagePatron,this), Packs2(40, 60,ImagePatron,this),
               PatronPackCount(80,60-5,200,this),   HitsCounter(160-
25,120-5,ImageTarget,0,30,this,-5)

                          {
                          PatronCount.Value.Font=COUNTER_FONT;
                          PatronCount.Value.SetScale(
COUNTER_SCALE);          //*************************
     ……..
     }
     };
     class TStatusBar1 : public TStatusBar
     {    public:
          TLabeledCounter ID;
          TIntegerText HID;
          TPicture Radio;
          TAccumStatus AccumulatorProgress;
        TAccumStatus AccumulatorHeadProgress;
          TStatusBar1(TScreenComponent *AOwner):TStatusBar(AOwner),
          ID(5,5,"ID ",WHITE,this->BColor, 123, 10  ,  this),
          HID(100,5,231,WHITE,this->BColor, this),
          Radio( this->Width-150,5,ImageWiFi_blue,this),
          AccumulatorProgress(this->Width-50,this),
          AccumulatorHeadProgress(this->Width-100,this)
          {
               AccumulatorProgress.Label='T';
               AccumulatorHeadProgress.Label='H';
               ID.SetFont(Courier_NewT16x32,1.0,4);
          HID.Font=Courier_NewT16x32;
               HID.font_thickness= 4;
          }
     };
     class TFightScreen:public TScreen
     {
     public:
          uint16_t backcolor;
          TStatusBar1 StatusBar;
          TPanel1 Panel1;
     TFightScreen(uint16_t    bcolor):TScreen(),    StatusBar(this),
Panel1(0,50,bcolor,this)
          {
               backcolor=bcolor;
               LCD_direction(3);        }
      void     SetAccumulator(float                          percent)
{StatusBar.AccumulatorProgress.SetProgress(percent);}
```

```
      void     SetHeadAccumulator(float                    percent)
{StatusBar.AccumulatorHeadProgress.SetProgress(percent);}

      void SetRadioVisible(bool v){StatusBar.Radio.SetVisible(v); }
      void RadioOFF(){SetRadioVisible(false);}
      void RadioON(){ SetRadioVisible(true);     }

      void SetHDAccVisible(bool v){StatusBar.HID.SetVisible(v);}
      void HeadSensorVisible(bool v)
      {
           SetRadioVisible(v);
           SetHDAccVisible(v);
      }    };
```

Екран режиму бою налаштовує свої фрагменті у своєму конструкторі: StatusBar – це верхня стрічка, та Panel1 – це решта екрану, яка відповідна за його зміст. Тобто щоб щось додати потрібно агрегувати компонент в клас екрана, а потім у конструктори задати його параметри. Крім того якщо потрібно змінити вигляд компонента, то можливо використовувати успадкування, щоб, наприклад, додати уже в нього які небуть елементи. Так, накштальт, зроблено у TStatusBar1 – були додані індикатори AccumulatorProgress, AccumulatorHeadProgress та ще деякі. Цей метод дуже схожий на тій що використовує Delphi. Можливо надалі на базі цієї бібліотеці компонент розробить деяку середу візуального проектування LCD-інтерфейсів.

5.4 Проектування бібліотеці програвання звукових ефектів

Для озвучування процесу гри була розроблена бібліотека призначена для використання в програмах мовою C++ для мікроконтролера STM32F407RG та сумісних, з обсягом вбудованої flash пам'яті не менш 1 МБ (див. лістинг Б.9). Вона реалізує відтворення стереофонічних звукових ефектів через вбудований ЦАП. Бібліотека розрахована на те, що звукові ефекти на зберігаються в flash пам'яті мікроконтролера, й описує структури для зберігання й доступу до них.

Звукові ефекти зберігаються у вигляді масиву байт, якому передує структура директорії звукових ефектів. Структури наведені у лістингах 5.4-5.6.

Лістинг 5.4 – Директорія звукових ефектів

```
typedef struct
{    uint32_t Index;//номер байта в масиві звуків
     uint8_t Compression;//  незжатий/зжатий
     uint32_t length; //довжина эфекта в байтах
}Tsounddirectoryentry;
typedef struct
{
     uint16_t Soundnum; // кількість ефектів усього
     Tsounddirectoryentry *Entries }Tsounddirectory;
const
Tsoundsheme*Soundsheme=(Tsoundsheme*)SOUNDSHEMEBEGIN_ADDRESS;
const
Tsounddirectory*Sounddirectory=(Tsounddirectory*)EFFECTSBEGIN
_ADDRESS;
```

А також існує структура – таблиця ефектів (див. лістинг 5.5), яка складається з декількох описів звукових ефектів (див. лістинг 5.6).

Лістинг 5.5 – Таблиця звукових ефектів

```
typedef struct
{
     uint8_t Nsounds;
     bool repeat;
     int8_t nexttable;//безумовний перехід на наступну таблицю
     Tsoundtableentry Sound[Soundtableentrynumber];
}Tsoundtable;
```

Лістинг 5.6 – Опис звукового ефекта

```
typedef struct      //опис звукового ефекту
          {              uint16_t  soundindex;//номер  елемента  в
Tsounddirectory->
               //sound properties
               uint8_t volume; //255 - max
               uint16_t playspeed;//коеф для перерахування швидкості
- чим більше тем швидше
               bool  timeclip;//якщо  true  -  playspeed  показує  не
швидкість а зменшення часу відтворення
               bool repeated; //повторювати безупинно
               bool nowaitnext;//не чекати наступного
} Tsoundtableentry;
```

Таблиця ефектів може запустити послідовно або одночасно кілька звукових ефектів, описаних у ній. Також таблиця може бути відтворена асинхронно – якщо всі її ефекти асинхронні. Якщо nexttable>=0, то відбувається перехід на таблицю nexttable.

Бібліотека складається з наступних файлів: Baseaudioclass.cpp, Baseaudioclass.h, Mtsound.cpp, Mtsound.h,Soundtables.cpp, Soundtables.h,types.h.

Щоб використовувати бібліотеку в програмі необхідно:

– занести оцифровані звукові ефекти й таблиці ефектів на згадку програм мікроконтролера за допомогою програматора;

– задати у файлі Soundtables.h адреси, відредагувавши рядка

#define SOUNDSHEMEBEGIN_ADDRESS 0x8004000;

– а також для початку таблиць ефектів:

#define EFFECTSBEGIN_ADDRESS 0x8020000;

– для адреси початку масиву ефектів і вставити директиву:

#include " Soundtables.h";

– викликати функцію InitSoundTables().

Для того щоб запустити відтворення таблиці слід викликати функцію void Playtable(int ntable), де ntable – номер таблиці. А також:

– void SoundTableTerminate(int ntable) – зупиняє відтворення таблиці;

– void AllToundTableTerminate() – зупиняє програвання усіх звуків;

– void Dochangeeffects(uint16_t modulv,int16_t a) – змінює швидкість відтворення на modulv і передає параметр a.

Для редагування таблиць був задіяний спеціально розробленій здобуток SEEditor на Delphi CE, форма якого подана на рисунку 5.6, який генерує файл прошивки у якому розташовані таблиці схем и ефектів.

Здобуток SEEditor здатен отримувати сторонні звукови файли, опрацьовувати и збирати у схеми.

Також генеруется файл дефініцій EVENTS.h до цих таблиц (див. лістинг 5.7)

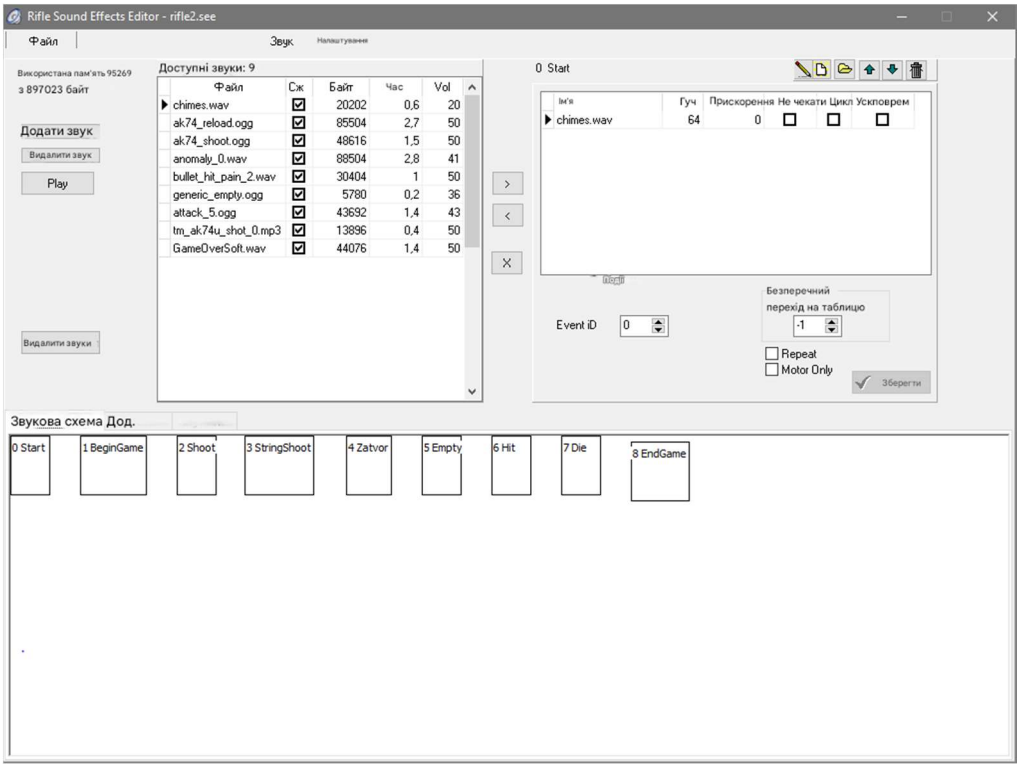

Рисунок 5.6 – SEEditor з завантаженої схемою для зброї

Листинг 5.7 – Константи з номерами подій

```
#ifndef EVENTS_h
#define EVENTS_h

#define START_EVENT 0
#define  BEGIN_EVENT 1
#define  SHOOT_EVENT 2
#define  STRINGSHOOT_EVENT 3
#define  RELOAD_EVENT 4
#define  EMPTY_EVENT 5
#define  HIT_EVENT 6
#define  DIE_EVENT 7
#define  ENDGAME__EVENT 8
#endif
```

Такім чином, щоб імітувати звук пострілу, потрібно виконати лише Playtable(SHOOT_EVENT). При цьому звук програватимуся асинхронно, що дуже важливо для подієвий моделі ПЗ.

5.5 Розробка інтерфейсу керування базовою станцією

Базова станція приєднується до ноутбука через перехідник USB2COM. Алгоритм роботи програмного забезпечення базової станції заснований на обробці переривань від її внутрішнього USART і переривання від модуля NRF.

Алгоритм обробки переривання від її внутрішнього USART по прийманню (від ПК) виглядає в такий спосіб.

Крок 1. Завантажить байт із буферного регістру USART.

Крок 2. Якщо байт перший у серії, то трактувати його як наказ змінити канал.

Крок 3. Якщо байт другий, то трактувати його як довжину посилки.

Крок 4. Якщо номер байта більше чому другий, то передати байт в NRF, зменшити довжину посилки.

Алгоритм обробки переривання від її модуля NRF по прийманню (від гвинтівки) виглядає в такий спосіб.

Крок 1. Прийняти пакет від NRF.

Крок 2. Передати номер гвинтівки, від якої він прийшов.

Крок 3. Передати пакет по USART на ПК.

Головна форма додатку зображена на рис. 5.7. Вона дозволяє ініціювати гру якщо натиснути кнопку «Старт раунд», або зупинити кнопкою «Стоп раунд». Кнопка «Параметри гри» відкриває діалогове вікно та дозволяє встановити та передати усім гвинтівкам параметри гри та зброї (див. рис. 5.8). Кнопка «Параметри зброї» відкриває діалогове вікно та дозволяє встановити та передати вказаної гвинтівці параметри зброї (див. рис. 5.9).

Кнопка «Звіт» примушує усі гвинтівки передати статистику до базової станції, після чого будується форма звіту зі статистикою бою (див. рис. 5.10).

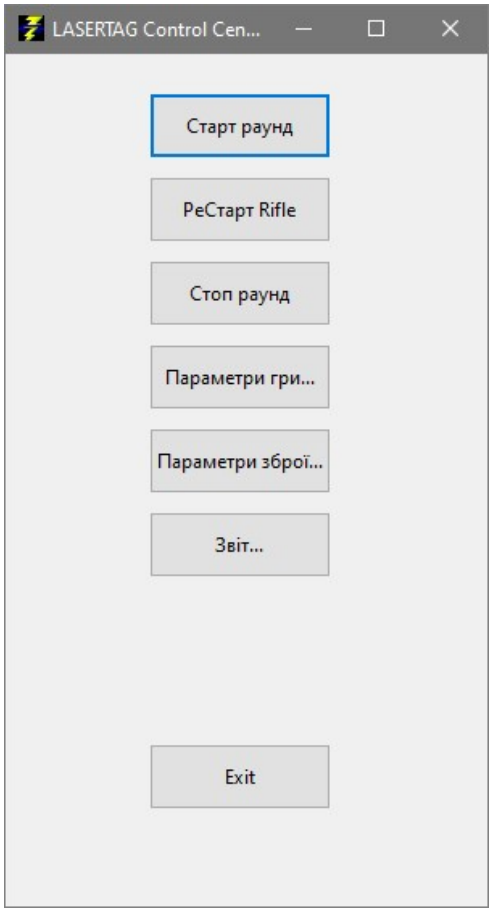

Рисунок 5.7 – Вигляд додатку для керуванням базової станцією

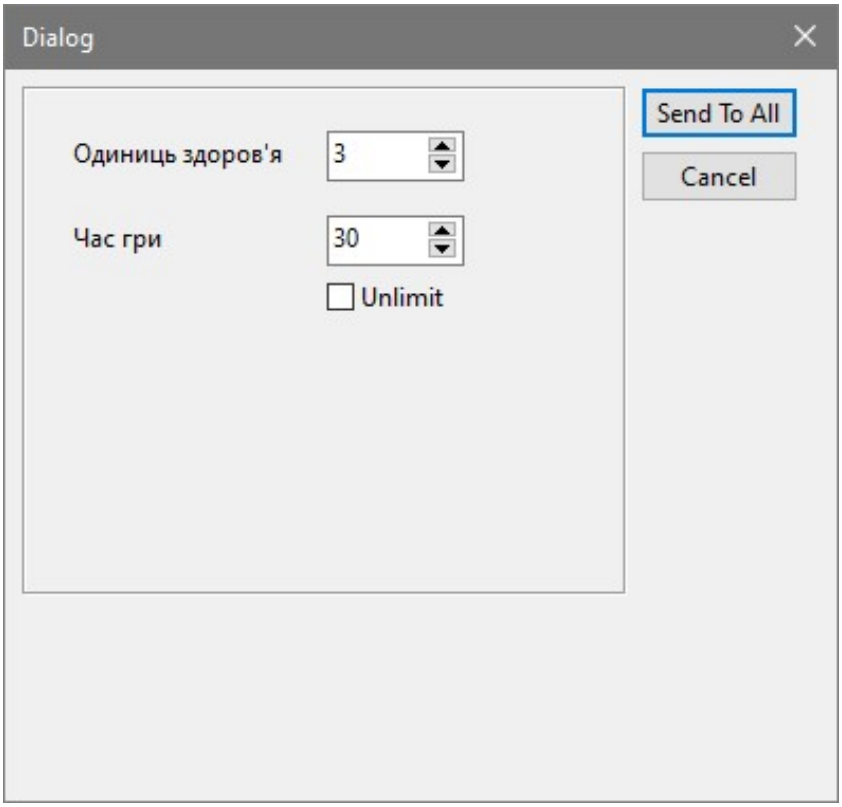

Рисунок 5.8 – Діалог параметрів гри

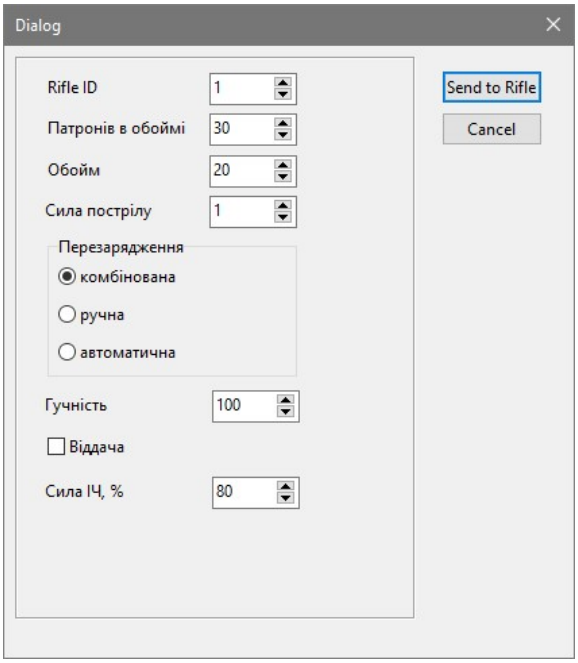

Рисунок 5.9 – Параметри зброї

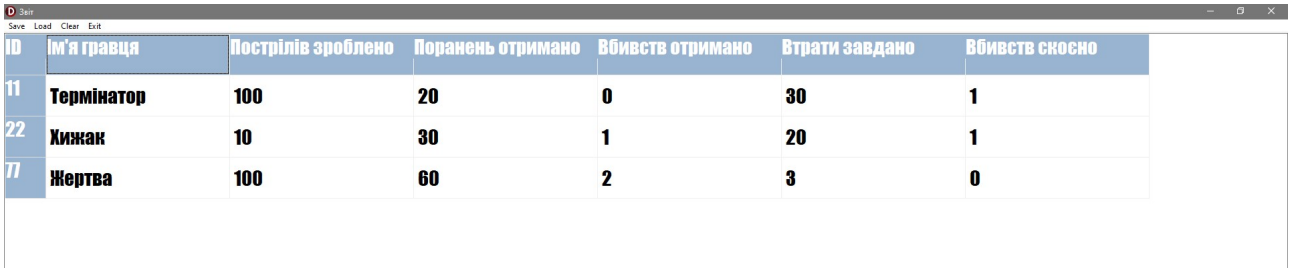

Рисунок 5.10 – Форма звіту зі статистикою бою

Таким чином розроблен здобуток для управління системою імітації бою.

5.6 Тестування системи

Для цілей тестування системи було проведено низку експериментів. Умови експерименту такі: гвинтівка оснащена ІЧ світлодіодом TSAL6100 (яскравість 1000мВт при піковому струмі 1А) без фокусуючої лінзи (кут розсіювання 10 градусів) обстрілювала малорухливу мету, забезпечену головними датчиками на основі TSOP572. Здійснювалося по 2000 пострілів на різній відстані та

фіксувалася кількість влучень. Потім шпаруватість модуляції світлодіода змінювалася, і серія дослідів повторювалася.

Перша серія експериментів проводилася при шпаруватості 10 відсотків, що відповідає 100мВт потужності. Результати наведені у табл. 5.2 та на рис. 5.11. З результатів видно, що зона впевненого прийому становить до 50 метрів, а при вже 75-80 метрах прийом припиняється).

Друга серія експериментів проводилася за шпаруватості 20 відсотків, що відповідає 200мВт потужності. Результати наведені у табл. 5.3 та на рис. 5.12. З результатів видно, що зона впевненого прийому становить до 65 метрів, а понад 80 метрів прийом різко погіршується.

Таблиця 5.2 – Залежність попадань від відстані при потужності 100мВт

Відстань, м	Кількість попадань	Кількість промахів
10	2000	0
15	2000	0
20	2000	0
25	2000	0
30	2000	0
35	2000	0
40	2000	0
45	2000	0
50	2000	0
55	1690	309
60	1350	649
65	973	1026
70	673	1325
73	300	1700
75	147	1853
80	0	2000

Третя серія експериментів проводилася за шпаруватості 80 відсотків, що відповідає 800мВт. Результати наведені у табл. 5.4 та на рис. 5.13. З результатів видно, що зона впевненого прийому зросла до 125 метрів, а понад 190 метрів прийом різко погіршується. И на прі кінець, четверта серія експериментів проводилася за максимальною шпаруватістю 99 відсотків, що відповідає приблизно1000мВт. Зрозуміло, що це екстремальні умови для світлодіода, і їх не потрібно постійно використовувати, якщо бажано продовжити життя світлодіодові. Результати наведені у табл. 5.5 та на рис. 5.14.

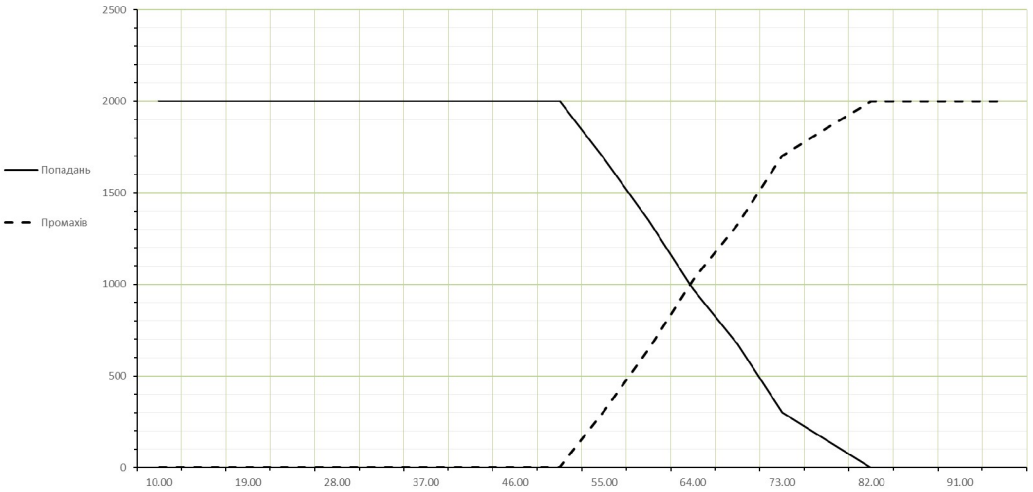

Рисунок 5.11– Графік залежністі попадань від відстані при потужності 100мВт

Таблиця 5.3 – Залежність попадань від відстані при потужності 200мВт

Відстань, м	Кількість попадань	Кількість промахів
10	2000	0
19	2000	0
25	2000	0
30	2000	0
40	2000	0
45	2000	0
50	2000	0
55	2000	0
60	2000	0
65	1993	1
70	1670	329
91	288	1710
95	155	1844
100	0	2000

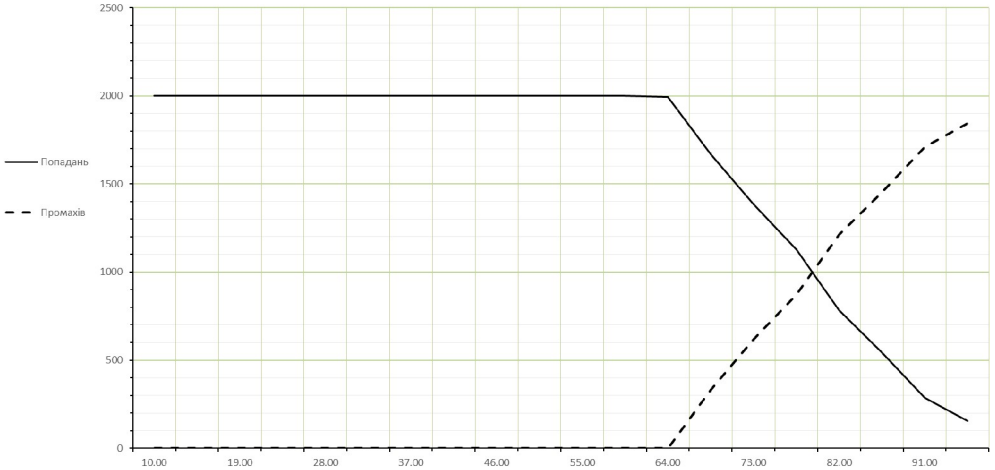

Рисунок 5.12– Графік залежністі попадань від відстані при потужності 200мВт

Таблиця 5.4 – Залежність попадань від відстані при потужності 800мВт

Відстань, м	Кількість попадань	Кількість промахів
10	2000	0
20	2000	0
30	2000	0
40	2000	0
50	2000	0
60	2000	0
70	2000	0
75	2000	0
85	2000	0
95	2000	0
105	2000	0
115	2000	0
125	2000	0
135	1789	208
145	1493	506
150	1201	799
160	873	1127
170	580	1418
180	300	1700
190	158	1842

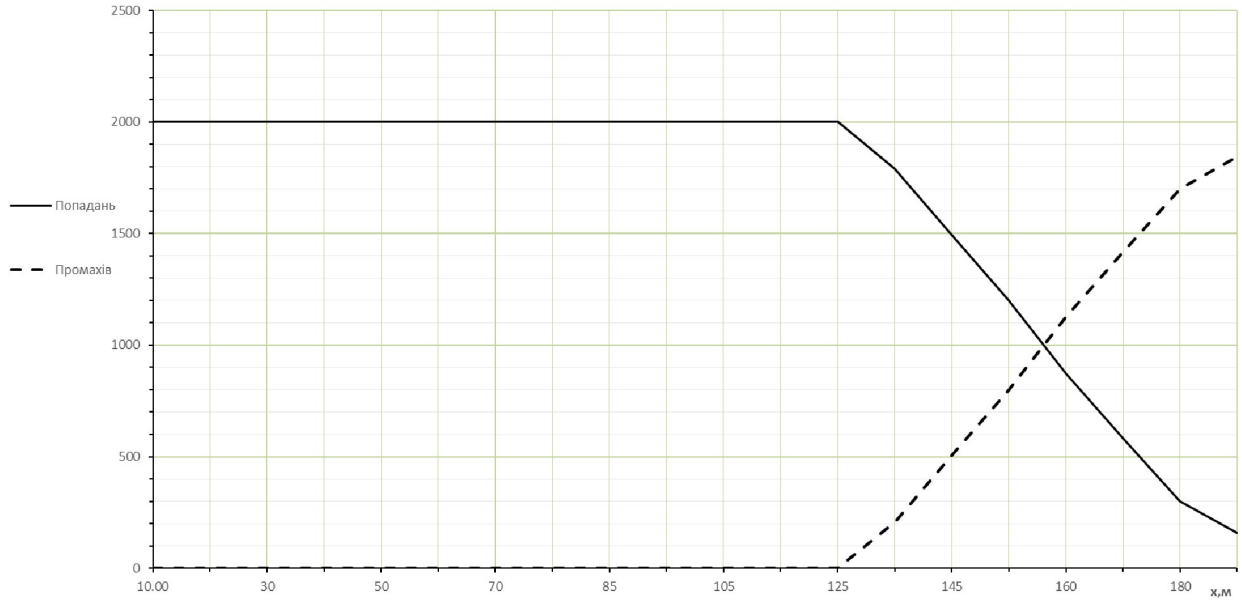

Рисунок 5.13– Графік залежністі попадань від відстані прі потужності 800мВт

З результатів видно, що зона впевненого прийому зросла приблизно до 150 метрів, а понад 190 метрів прийом погіршується, але ще можливо влучати у ціль десь на дистанції 200 метрів. Це максимально комфортна дистанція прицілювання – щоб надійно влучати треба ретельно прицілюватися.

Таблиця 5.5 – Залежність попадань від відстані прі потужності 990мВт

Відстань, м	Кількість попадань	Кількість промахів
10	2000	0
20	2000	0
30	2000	0
40	2000	0
50	2000	0
60	2000	0
70	2000	0
75	2000	0
85	2000	0
95	2000	0
105	2000	0
115	2000	0
125	2000	0
135	2000	0
145	2000	0
150	1693	306
160	1424	573
170	1154	845
180	870	1129
190	651	1347

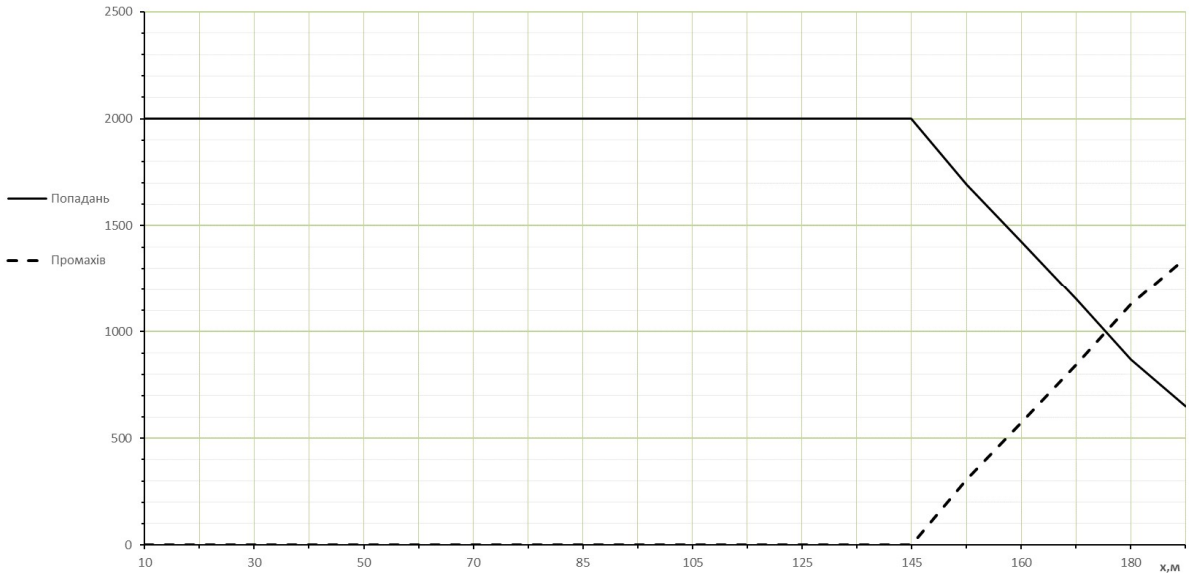

Рисунок 5.14– Графік залежністі попадань від відстані прі потужності 990мВт

До речі – більша кількість випадків втрачених пакетів пов’язана як раз з перериванням потіка даних за рахунок тремтіння ствола та рухові цілі. Ще раз наголошуємо, що ці результати були отримані без будь-яких лінз, але це не означає, що ними не варто користуватися – сенс у тому, що світлова пляма від світлодіода може досягати більш 10 метрів на відстані 50 метрів. Лінза з кутом у 2-3 градуси дає пляму два-три метра у діаметрі. Але ж є ще один аспект – це

використання лазертага у приміщеннях. Якщо влаштувати дуель у спортзалі, то 10 відсотків може бути забагато за рахунок відбитків променій від стін. Тобто регуляція потужності світлодіода є досить актуальною. Що до практичних застосунків – отримані результати виглядають дуже добре и дозволяють використовувати розроблену систему.

ВИСНОВКИ

У дипломній роботі було здійснено комплексну розробку ієрархічної системи с використанням технології РЧ-зв’язку. Проект охоплює аналіз, проектування та розробку програмної та апаратної частини системи.

У апаратної частини роботи був спроектований універсальний модуль для гри в Lasertag. Його можливо інтегрувати у модель зброї або пульту керування грою. У програмної частини било спроектована об’єктна модель ПЗ системи та вирішені наступні завдання: кодування та передачу ІЧ сигналу з виправленням помилок; відображення інтерфейса з допомогою бібліотеці класів, асинхронне подієво-орієнтоване відтворення звукових ефектів гри.

Наукова новизна полягає в розробці комплексного рішення що до проблем стійкості системи імітації бою в умовах перешкод, та автономності гравця, що забезпечує більше можливостей що до реалізації сценаріїв стрілецького бою.

Практична вартість розробленої системи полягає в її здатності служити як комерційним цілям так і тренування військових за умов наближених до реальних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Комп'ютерна програма "Програма для мікропроцесорної навчальної системи імітації стрілецької зброї шляхом передачі інфрачервоного сигналу": свідоцтво Україна № 65680; заявл. 26.05.2016; опубл. 29.07.2016, Бюл. № 41. 22 с.

2 Комп'ютерна програма "Програма для інфрачервоного пульта керування навчальною системою імітації стрілецької зброї": свідоцтво Україна № 65681; заявл. 26.05.2016; опубл. 29.07.2016, Бюл. № 41. 22 с.

3 Комп’ютерна програма "Бібліотека для програвання звукових ефектів для мікроконтролерів STM32F4XX": свідоцтво Україна № 95590; заявл. 23.01.2020; опубл. 23.01.2020, Бюл. № 57. 25 с.

4 Об'єктно-орієнтований підхід до дизайну інтерфейсів мікропроцесорних пристроїв [Електронний ресурс] / Сгадов С. О., Тур'янський Е.Д. // Тиждень науки: щоріч. наук.- практ. конф., 13–17 квітня 2020 р.: тези доп. / Редкол.: В.В. Наумик (відпов. ред.) Електрон. дані.- Запоріжжя : НУ«Запорізька політехніка», 2020. - С. 70-72

5 Conforti, Gilbert. Dynamics Through Devices. Infantry. 1973. Vol. 63, no. 1. P. 13. URL: https://babel.hathitrust.org/cgi/pt

6 Bulova Annual Report to the stockholders 1959/60. (англ.) *NY: Bulova Watch Company.* 1960. - P. 14 - 18 p.

7 Multiple Integrated Laser Engagement System (MILES). URL: https://man.fas.org/dod-101/sys/land/miles.htm.

8 Wireless Sensor Networks with LoRaWAN and Wi-Fi.. IEEE Transactions on Industrial Informatics, 2019. 15 (4), 1713-1722. DOI: 10.1109/TII.2018.2874455

9 ESP8266 Overview. URL: www.espressif.com/en/products/socs/esp8266

10 ESP32 Datasheet. URL: *https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf*

11 Нікольський Ю. В., Пасічник В., Щербина Ю. Дискретна математика. К. : Вид. група BHV, 2007. 368 с.

12 Richard W. Hamming. Error Detection and Error Correction Codes. *The Bell System Technical Journal*. 1950. Vol. XXIX, no. 2. P. 147–160.

13 Semerenko V. Parallel Cyclic Codes. *Вісник Вінницького політехнічного інституту.* 2014. No. 6.

14 Gokkengam A. Codes correcteurs d'erreurs. *Chiffres*. 1959. Vol. 2. P. 147–156.

15 Матов О. Я., Василенко В. С. Блокові згорткові коди в задачах контролю цілісності. *Реєстрація, зберігання і обробка даних*. 2007. Т. 9, № 1. С. 100–107.

16 Declercq D., Fossorier M. Decoding algorithms for nonbinary LDPC codes over GF(q). *IEEE transactions on communications*. 2007. Vol. 55, no. 4. P. 633–643.

17 Chen J. Reduced-complexity decoding of LDPC codes. *IEEE transactions on communications*. 2005. Vol. 53, no. 8. P. 1288-1299

18 Berrou C., Glavieux A., Thitmajshima P. Near Shannon Limit error-correcting coding and decoding: Turbo-codes. *IEEE transactions on communications*. 1996. Vol. 44, no. 8. P. 1261-1271

19 Слюсар В. И. Синтез LDPC и полярных кодов на основе торцевого произведения матриц. *Розвиток освіти, науки та бізнесу: результати 2020: тези доп. міжнародної науково-практичної інтернет-конференції*, 3 - 4 грудня 2020 р. - Україна, Дніпро, 2020. - Т.2. - С. 393-396.

20 Arikan E. Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE transactions on communications*. 2009. Vol. 55, no. 8. P. 3051-3073 Mahfouz M. Z., Kokkeler A. B. J., Meijerink A., Alayón Glazunov A. Impact of Ultra-Narrowband Interference on Wi-Fi Links: An Experimental Study // IEEE Transactions on Wireless Communications. 2021. Vol. 20, No. 5. P. 1-12. URL: https://ris.utwente.nl/ws/portalfiles/portal/263618455/Mahfouz2021impact.pdf

21 FORPOST Що таке лазертаг.
URL: https://lasertag.kharkov.ua/download/about_forpost_ua.pdf

22 IEEE 802.15.4 Performance with Wi-Fi Interference // Scientific.Net. 2014. Vol. 526. P. 330-335. URL: https://www.scientific.net/AMM.526.330

23 Experimental Assessment of the Effects of Building Materials on Wi-Fi // Scientific Research Publishing. 2022. URL: https://www.scirp.org/journal/paperinformation?paperid=133136

24 Wireless Interference Analysis for Home IoT Security Vulnerability // Ulster University Research Portal, 2022. URL: https://pure.ulster.ac.uk/ws/portalfiles/portal/91482170/A_Novel_Approach_to_Home_IoT_Vulnerability_Detection_Cryptography_v3_1_col_anon_SHORT.pdf

25 Radio Interference of Wireless Networks and the Impact of AR/VR Applications // MDPI Electronics Journal, 2022. URL: https://www.mdpi.com/2079-9292/12/1/67

26 Finite Field Arithmetic and Reed-Solomon Coding URL: https://research.swtch.com/field.

ДОДАТОК А

ЕЛЕКТРИЧНІ СХЕМИ

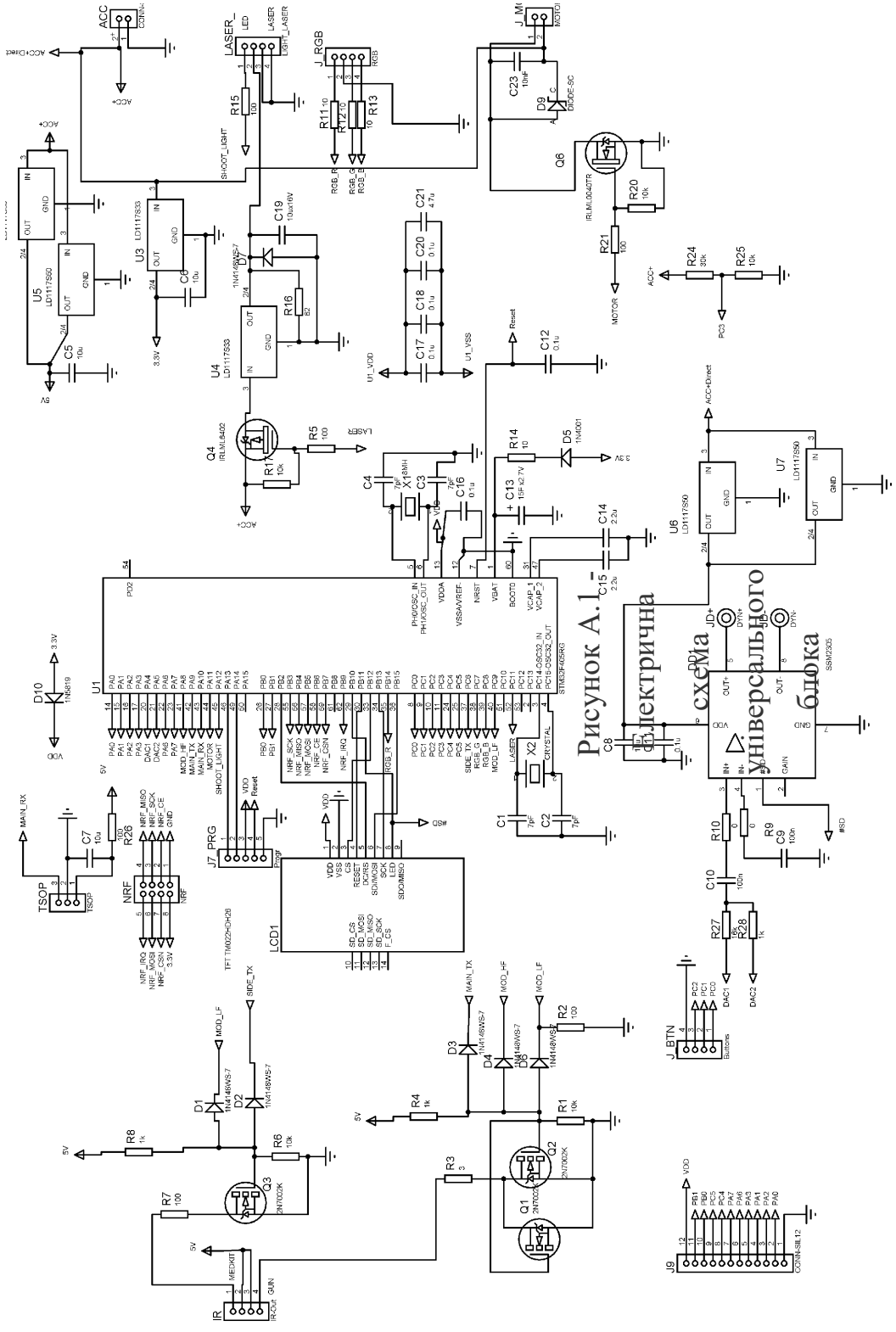

Рисунок А.1 - Електрична схема універсального блока

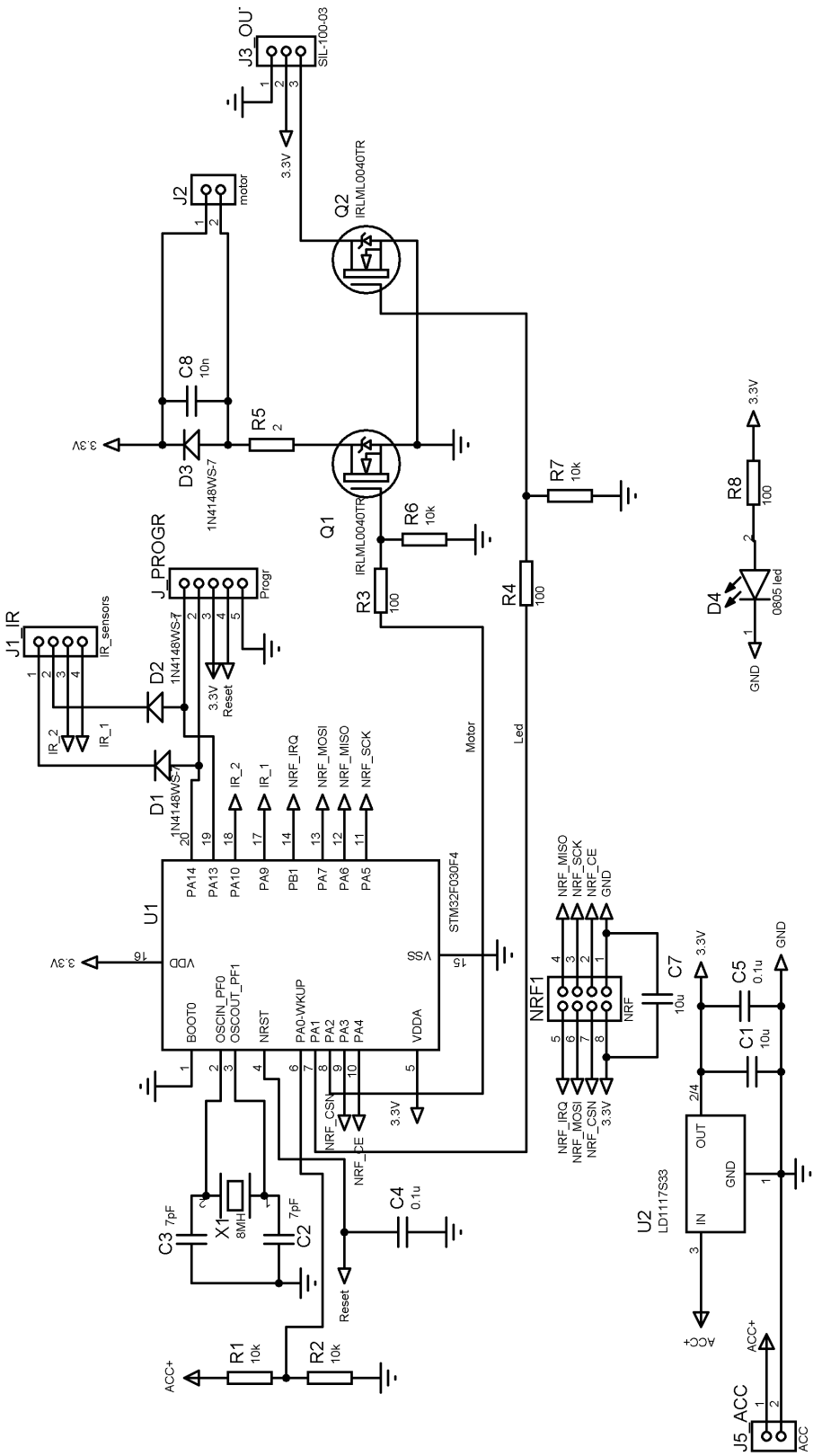

Рисунок А.2 - Електрична схема мікропроцесорного блока датчика влучень

ДОДАТОК Б
ФРАГМЕНТИ КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

Лістинг Б.1 – Програма гвінтівки

```
#include "systick.h"
#include <stm32f4xx.h>
#include <stdbool.h>
#include "TRifle.h"
#include <stdint.h>
#include "delay.h"
#include "sys.h"
#include "lcd.h"
TRifle *r;
int main(void)
{
  r=new TRifle();
  r->Run();
  return 0;
  }
```

Лістинг Б.2 - Класи `TRifle` та `TTask`

```
#ifndef TRIFLE_H
#define TRIFLE_H

#include "TTaskshell.h"
#include "baseclass.h"
#include "rifleconfig.h"

class TTask;
//основний цикл програми
class TRifle : public TTaskShell
{
   public:
        TConfig config;
    TRifle();
    virtual ~TRifle();
    virtual void InitDrivers();
    virtual void Init();

      bool CheckSeviceMode();
    void  DoReset();
```

```
        void OnSendConfig();
            void  Shoot();
            void  Recharge();
            bool KeyPressed(int keycode);
      protected:


    };

    class TTask : public TBaseAppClass{
      protected:
        TRifle *rifle;
        virtual void InitDrivers(){};//optional
          virtual void InitDisplay(){};
      public:
      TTask(TRifle *AOwner){rifle=AOwner;};

        virtual void Init(){
             InitDisplay(); InitDrivers();
        };
    };
#endif
```

Лістинг Б.2 – Клас TTaskShell

```
    #ifndef TTASKSHELL_H
    #define TTASKSHELL_H
    #include "baseclass.h"
    #define NTASKS 2

    //task manager
    class TTaskShell :public TObject
    {   protected:
      unsigned int Count;
        TBaseAppClass *tasks[NTASKS] ;
      virtual void InitDrivers(){};

      public:
      TTaskShell( ){   Count=0;
       for(int i=0;i<NTASKS;i++)tasks[i]=NULL;
       };
      virtual     ~TTaskShell(){ };

         virtual void Init();

       void Run(){
          for(unsigned int i=0;i<Count;i++)
           if(tasks[i]!=NULL) StartTask(tasks[i]);
             };
      int AddTask(TBaseAppClass*task)  {
            if(task==NULL) return -1;
```

```
        if(Count<NTASKS) {    tasks[Count ++]=task;   return
    Count-1;}
                    else return -1;
      }    ;

       void StartTask(TBaseAppClass*task ){
       if(tasks==NULL) return;
            task->Init();
          task-> Run();
          while (!task->Terminated);
          task->DeInit();
      }
       void StartTask(int task ){
       if (task<Count)StartTask(tasks[task]);
      }

      void TerminateTask(TBaseAppClass*task ){
       if(tasks==NULL) return;
       task->Terminate();
      }
      void TerminateTask(int task ){
              if (task<Count)TerminateTask(tasks[task]);
       }
      void Clear() {
         for(int i=0;i<Count;i++){
           if(tasks[i]!=NULL) TerminateTask(tasks[i])    ;
           tasks[i]=NULL;
       }
       Count=0;
      }
    };

#endif
```

Лістинг Б.3 – Клас `TBaseAppClass`

```
    #ifndef  baseclass_h
    #define baseclass_h
    #include "Messages.h"

    class TObject
    {
      public:

    TObject(){};

    };


    class TBaseAppClass:public TObject
    {
      protected :
```

```
        virtual bool GetMessage(TMessage &Message);
      public:
          bool         Terminated;
        virtual void Terminate(){Terminated=true;};
        virtual void Dispatch(TMessage Message){};
        virtual void OnMessage(TMessage Message){};
        virtual void OnCPUIdle(){};
        virtual void Init(){};
          virtual void DeInit(){};
        virtual void Run();
      TBaseAppClass():TObject(){Terminated=false;}

      virtual~TBaseAppClass(){   };
    };


#endif
```

Лістинг Б.4 – Клас режиму очікування.

```
    #ifndef TDEFAULTMODEAPP_H
    #define TDEFAULTMODEAPP_H

    #include "TRifle.h"
    #include "defaultscreen.h"

    class TDefaultModeApp : public TTask
    {
      public:

        TDefaultModeApp(TRifle* AOwner):TTask(AOwner){};
      virtual     ~TDefaultModeApp(){

        };

        virtual void Dispatch(TMessage Message);
      protected:
        DEFAULT_MODE::TDefaultScreen* screen;
       virtual void InitDisplay();
      void OnUnknown(){};
      void OnRifleFireme(){};
      void DoStartGame(){

        Terminate();
      }
      void DoReset();
      void OnSendConfig();
    };

#endif
```

Лістинг Б.5 – Клас режиму бою.

```
#ifndef TFIGHTMODEAPP_H
#define TFIGHTMODEAPP_H

#include "TRifle.h"
#include "fightscreen.h"

class TFightModeApp : public TTask
{
  public:
         TFightModeApp(TRifle* AOwner):TTask(AOwner){
         zatvor=false;
         trigger=false;
         };
    virtual ~TFightModeApp(){
         DeInit();
    }


  virtual void Dispatch(TMessage Message);
  virtual void OnCPUIdle();

  void OnTriggerKeyPressed();
  void OnShutterKeyPressed() ;
  protected:
  void   DoFinishGame();
  void   OnRifleFireme(){
    DoNRFSendHit();
    DoSoundHit();
  };
  void   DoSendAcknStatus();
  void OnUnknown(){};

  protected:
  bool trigger;
  bool zatvor;
  TFightScreen* screen;
  void DoNRFSendHit();
  void DoIncreaseMyHits();
  void DoRefreshDisplay()    ;
  virtual void InitDisplay();
  };

#endif
```

Лістинг Б.6 – Програма датчіку влучення.

```
#include "_delay.h"
#include <stm32f0xx_gpio.h>
#include <stm32f0xx_rcc.h>
#include <stm32f0xx_spi.h>
#include "pindefsines.h"
#include "Transivers.h"
#include "Messages.h"
#include "HSconfig.h"
#include "Voltcontrol.h"
#pragma pack(1)

#include "nrf24.h"

#include "nrfpayload.h"


uint32_t i,j,k;
uint16_t m;

TEEPROM Eprom;

void AfterRXRead(uint8_t* _payload,uint8_t length)
 {
   if (length==0) return;
   TMessage M;
   for(int i=0;i<4;i++) M.bytes[i]=_payload[i];
   // покладемо прийняте в кільцевий буфер
   irbuf.Write(M.data);
   NRFBlock_IRQRX_Clear();
//   nRF24_Switch_TX_Mode();
 }
nRF24_TXResult tx_res;
int main(void) {

  SystemInit();
   InitVoltControl();
   InitIRReciver();
   configNRFTX(115,nRF24_PIPETX,4);
   NRFBlock_IRQRX_init(AfterRXRead);
   configNRFRX(115,nRF24_PIPE1,4);

nRF24_Switch_RX_Mode();

   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);

GPIO_InitTypeDef PORT;
    //LED init
    PORT.GPIO_Mode = GPIO_Mode_OUT;
  PORT.GPIO_Speed = GPIO_Speed_Level_1;
  PORT.GPIO_OType = GPIO_OType_PP;
```

```
   PORT.GPIO_PuPd = GPIO_PuPd_NOPULL;
   PORT.GPIO_Pin   = MOTOR|LED;
      GPIO_Init(GPIOA, &PORT);

   PORT.GPIO_Mode = GPIO_Mode_IN;
   PORT.GPIO_Speed = GPIO_Speed_Level_1;
   PORT.GPIO_OType = GPIO_OType_PP;
   PORT.GPIO_PuPd = GPIO_PuPd_NOPULL;
  PORT.GPIO_Pin   =IR1|IR2|IR3|IR4;
  GPIO_Init(GPIOA, &PORT);
     // Initialize delay
     Delay_Init();


   while (1)
     {
      //TSIRCData data;
      TMessage  M;
     if(!irbuf.IsEmpty())
     {
       while (irbuf.Read(M.data))
          {
                              uint8_t command_type=M.Msg.cmd |
0x0f0;
               if(command_type==ACTION)
               {
                    if(M.Msg.cmd==ACTION_LED)
                    {
                     GPIO_WriteBit(GPIOA,
   LED,(M.Msg.Arg1!=0?Bit_SET:Bit_RESET));
                    continue;
                    }
                    if(M.Msg.cmd==ACTION_MOTOR)
                    {
                         GPIO_WriteBit(GPIOA,
   MOTOR,(M.Msg.Arg1!=0?Bit_SET:Bit_RESET));
                         continue;
                    }
                    if(M.Msg.cmd==ACTION_ASK_STATUS)
                    {
                    nRF24_Switch_TX_Mode();
                    Msg_Clear(M);
          M.Msg.Target=0;
                    M.Msg.cmd=RIFLE_STATUS;
          M.Msg.Arg1=GetUsuplyPercent();

           tx_res = nRF24_TransmitPacket(M.bytes, 4);
           nRF24_Switch_RX_Mode();
                    continue;
                    }
                    if(M.Msg.cmd==ACTION_ASSIGN_ID)
                    {
```

```
        if((M.Msg.Target^M.Msg.Arg1^M.Msg.cmd^0x12)==M.Msg.Arg2)
      {//control sum check
                              if(M.Msg.Target==M.Msg.Arg1){
                                   Eprom[SENSOR_ID]=M.Msg.Target;
                                   for(int m=0;m<5;m++)
                                   {
                                    GPIO_WriteBit(GPIOA,
        LED,Bit_SET);
                                        _delay_ms(250);
                                    GPIO_WriteBit(GPIOA,
        LED,Bit_RESET);
                                        _delay_ms(250);
                                   }
                              }
                         }
                              continue;
                         }
                    }
               else
                    {
                                   nRF24_Switch_TX_Mode();
                      //  nRF24_CE_L();
                     //configNRFTX();
                         for(int i=0;i<10;i++)
                     {
                     tx_res = nRF24_TransmitPacket(M.bytes, 4);
                     nRF24_Switch_RX_Mode();
                     if(tx_res==nRF24_TX_SUCCESS)break;
                    }
                         }
                     }
           }
          }
      }
      #ifdef __cplusplus
       extern "C" {
      #endif

      void  NMI_Handler ()
      {
        while(1);
      }

      void HardFault_Handler()
      {
          __ASM volatile("BKPT #01");
        while(1);
      }
      #ifdef __cplusplus
       }
#endif
```

Лістинг Б.7 – Програма базової станції.

```
 #include "delay.h"
#include "sys.h"
#include "nrfpayload.h"
#include "Transmit.h"
#define TIMEOUT 1000
void AfterRXRead(uint8_t* _payload,uint8_t length)
 {
   if (length==0) return;
   nRF24_SetRFChannel (payload[0]);
   TMessage M;
   for(int i=2;i<payload[1];i++) M.bytes[i]=_payload[i];
      irbuf.Write(M.data);
   NRFBlock_IRQRX_Clear();
   nRF24_Switch_TX_Mode();
nRF24_WritePayload(payload,length);
 }

  int j=3;
  uint8_t Buf[32];

void AfterRXHandler(uint8_t* payload,uint8_t length)
{

    NRFBlock_IRQRX_Clear();
  //     nRF24_Switch_TX_Mode();
USART_Send(payload, length)
                  nRF24_ClearIRQFlags();
  //                    nRF24_Switch_RX_Mode();
    }
}

int main(void)
{
  delay_init();

InitUARTTransmitter();

nRF24_Switch_RX_Mode();
   configNRFTX(115,nRF24_PIPETX,4);
   UART_IRQRX_init(AfterRXRead);
   NRFBlock_IRQRX_init(AfterRXHandler);
   configNRFRX(115,nRF24_PIPE1,4);

  while(1)
  {


  }
```

Лістинг Б.8 – Файл Soundtables.h

```
#ifndef  SoundTables_h
#define SoundTables_h
#include <stdint.h>
#include <stdbool.h>
#include "MTSound.h"
#define SoundTablePrimaryNumber 40
#define SoundTableSecondaryNumber 20
#define SoundTableRandomNumber 10
#define                                          SoundTableNumber
(SoundTablePrimaryNumber+SoundTableSecondaryNumber+SoundTable
RandomNumber)

#define SoundTableEntryNumber 23
#define SOUNDSHEMEBEGIN_ADDRESS 0x8020000
#define EFFECTSBEGIN_ADDRESS    0x8025000
//(SOUNDSHEMEBEGIN_ADDRESS+sizeof(TSoundSheme))
//0x801C000// 0x8004000
//0x8020000


#define STARTTABLE_Index 0
#define BEGIN_TABLE_Index 1
#define SHOOT_TABLE_Index 2
#define STRINGSHOOT_TABLE_Index 3
#define RELOAD_TABLE_Index 4
#define EMPTY_TABLE_Index 5
#define HIT_TABLE_Index 6
#define DIE_TABLE_Index 7
#define ENDGAME_TABLE_Index 8

#pragma pack(push,1)
//typedef uint16_t TSoundTableAddreses[SoundTableNumber];
typedef struct
    {
         bool enabled;
         int8_t value;
         int8_t toTable;
    } TCondition;

typedef struct
    {
         uint16_t soundindex
         uint8_t volume; //255 - max
         uint16_t playspeed;
         bool timeclip
         bool repeated;
         bool nowaitnext
    } TSoundTableEntry;
typedef struct
{
  uint8_t NSounds;
```

```
   uint8_t Event_Id;     bool repeat;
   bool Motoronly;
   int8_t nextTable TSoundTableEntry
Sound[SoundTableEntryNumber];
 }TSoundTable;

 typedef struct
     {
      uint8_t GeneralVolume;

       uint8_t  soundTablePrimaryNumber,
                  soundTableSecondaryNumber,
                  soundTableRandomNumber;

     TSoundTable SoundTables[SoundTableNumber];// містить три
                                                //набори таблиць
   } TSoundSheme;

 typedef struct
 {
   uint32_t Index;//номер байта
   uint8_t Compression;
   uint32_t length;
   uint16_t maxvalue;
 }TSoundDirectoryEntry;

 typedef struct
 {
     uint16_t SoundNum;
     TSoundDirectoryEntry Entries[4];
  }TSoundDirectory;
 #pragma pack(pop)

 extern const TSoundSheme *SoundSheme;
 extern const TSoundDirectory *SoundDirectory;
 void InitSoundTables();
 void AllSoundTableTerminate();
 void SoundTableTerminate(int nTable);
 void DoChangeEffects(uint16_t modulv,int16_t a);
 void PlayTable(int nTable);
 void PlaySecondaryTable(int nTable);
 void SoundTableWait(int nTable);
 bool SoundTableIsPlaying(int nTable);

  void *SoundEffectAddress(uint16_t soundindex);
 TSoundDescriptor PlayTableSound(int nTable,int i);
  void    PlayTable_Event    (uint8_t    Event_ID)    ;    int
PlaySingleTable_Event  (uint8_t Event_ID);
 void PlaySingleTable_Event_and_Wait  (uint8_t Event_ID);
 #endif
```

Лістинг Б.9 – Файл Soundtables.cpp

```
#include "SoundTables.h"
#include "MTSound.h"

const          TSoundSheme          *SoundSheme=(TSoundSheme
*)SOUNDSHEMEBEGIN_ADDRESS;
const    TSoundDirectory    *SoundDirectory=(TSoundDirectory
*)EFFECTSBEGIN_ADDRESS;//sector5-sector11

volatile int CurrentTableIndex;
volatile int CurrentSecondaryTableIndex;
volatile int CurrentRandomTableIndex;
void *SoundEffectAddress(uint16_t soundindex)
{
return (void *)((uint32_t) SoundDirectory +
         SoundDirectory-
>SoundNum*sizeof(TSoundDirectoryEntry)
         +sizeof(SoundDirectory->SoundNum)+
         +SoundDirectory->Entries[soundindex].Index-1);
}

//const
void InitSoundTables()
{
  CurrentTableIndex=-1;//  STARTTABLE_Index;
  CurrentSecondaryTableIndex=-1;
  CurrentRandomTableIndex=-1;
  InitSound();
}

TSoundDescriptor PlayTableSound(int nTable,int i);

void OnEndSoundPlay(TSoundDescriptor descr)
{
  int aCurrentTableIndex=descr.table;
  if (aCurrentTableIndex<0)return;
  TSoundTable  *  CurrentSTable=(TSoundTable  *)&(SoundSheme-
>SoundTables[aCurrentTableIndex]);
if (CurrentSTable==NULL)return;
int index=descr.index+1;

  if(index>=CurrentSTable->NSounds)
  {
    if (CurrentSTable->repeat) index=0;
    else {
              if((CurrentSTable-
>nextTable>=0)&&(CurrentSTable->nextTable<SoundTableNumber)
                        //&&!(CurrentSTable-
>vmin.enabled||CurrentSTable->vmax.enabled||
                        //CurrentSTable-
>amin.enabled||CurrentSTable->amax.enabled)
                    )
```

```
                    {
                         PlayTable(CurrentSTable->nextTable);
                         BreakPlaySoundThreads();
                    }


           return;
          } ;
    }
    for(  ;index<CurrentSTable->NSounds;index++)
    {
     PlayTableSound(aCurrentTableIndex,index);
     if(!CurrentSTable->Sound[index].nowaitnext   )break;
    }
  }

  void PlayTable(int nTable)
  {
  TSoundDescriptor ini;
  if (nTable<0)return;
  if (nTable>=SoundTablePrimaryNumber)return;

    ini.hdc=0;
    ini.index=-1;

  SoundTableTerminate(CurrentTableIndex);
  CurrentTableIndex=nTable;
  ini.table=CurrentTableIndex;
  UnBreakPlaySoundThreads();
  OnEndSoundPlay(ini);
  //CurrentTableIndex=-1;
  }

  void OnEndSoundPlay2(TSoundDescriptor descr)
  {
    int aCurrentTableIndex=descr.table;
    if (aCurrentTableIndex<0)return;
    TSoundTable   *   CurrentSTable=(TSoundTable   *)&SoundSheme-
>SoundTables[aCurrentTableIndex];
  if (CurrentSTable==NULL)return;
  int index=descr.index+1;

    if(index>=CurrentSTable->NSounds)
    {
      if (CurrentSTable->repeat) index=0;
      else {
                if((CurrentSTable-
>nextTable>=0)//&&(CurrentSTable->nextTable<SoundTableNumber)
                          //&&!(CurrentSTable-
>vmin.enabled||CurrentSTable->vmax.enabled||CurrentSTable-
>amin.enabled||CurrentSTable->amax.enabled)
                     )
                     {
```

```
                        PlaySecondaryTable(CurrentSTable-
>nextTable);
   BreakPlaySoundThreads();
                   }
                   else aCurrentTableIndex=-1;
          return;
         } ;
   }
   for(  ;index<CurrentSTable->NSounds;index++)
   {
    PlayTableSound(aCurrentTableIndex,index);
    if(!CurrentSTable->Sound[index].nowaitnext   )break;
   }
 }


 void PlaySecondaryTable(int nTable)
 {
   TSoundDescriptor ini;
   if (nTable<0)return;
   if (nTable>=SoundTableSecondaryNumber)return;
   nTable+=SoundTablePrimaryNumber;

   ini.hdc=0;
   ini.index=-1;
   CurrentSecondaryTableIndex=nTable;
 ini.table=nTable;
 OnEndSoundPlay2(ini);
 }

 void OnEndSoundPlayRand(TSoundDescriptor descr)
 {
   int aCurrentTableIndex=descr.table;
   if (aCurrentTableIndex<0)return;
   TSoundTable  *  CurrentSTable=(TSoundTable  *)&SoundSheme-
>SoundTables[aCurrentTableIndex];
 if (CurrentSTable==NULL)return;
 int index=descr.index+1;

   if(index>=CurrentSTable->NSounds)
   {
     if (CurrentSTable->repeat) index=0;
     else {
              if((CurrentSTable->nextTable>0)
                        //&&!(CurrentSTable-
>vmin.enabled||CurrentSTable->vmax.enabled||CurrentSTable-
>amin.enabled||CurrentSTable->amax.enabled)
                   )
                   {
                        PlaySecondaryTable(CurrentSTable-
>nextTable);//реурсивный переход к проигрыванию следующей
таблицы ,
```

```
                        BreakPlaySoundThreads();
                    }
          return;
         } ;
  }
  for(  ;index<CurrentSTable->NSounds;index++)
  {
   PlayTableSound(aCurrentTableIndex,index);
   if(!CurrentSTable->Sound[index].nowaitnext   )break;
  }
}

void PlayRandomTable()
{
  int nTable=0;//RND(SoundTableRandomNumber);
  nTable+=SoundTablePrimaryNumber+SoundTableSecondaryNumber;
  TSoundDescriptor ini;
  ini.hdc=0;
  ini.index=-1;
  CurrentRandomTableIndex=nTable;
  ini.table=nTable;
  OnEndSoundPlay(ini);
}
TSoundDescriptor PlayTableSound(int nTable,int i)
{
//if (nTable<0) return {-1,-1,-1};
//if (i<0) return {-1,-1,-1};
TSoundTableEntry Sound;
Sound=SoundSheme->SoundTables[nTable].Sound[i];
//float   volume = Sound.volume/127.0;
if(Sound.nowaitnext)
{
  return      MTPlaySound(nTable,i,
SoundDirectory->Entries[Sound.soundindex].Compression
                                        );
}
else
{
    return         MTPlaySoundEx(nTable,i,
                                        SoundDirectory-
>Entries[Sound.soundindex].Compression,
                                        OnEndSoundPlay);
}
}
void SoundTableTerminate(int nTable)
{
  if(nTable<0) return;
  TSoundThread*tmp=MT_FIRST;

  while(tmp!=NULL)
  {
              TSoundThread*tmp2=tmp->next;
```

```
                if(tmp->SoundDescriptor.table==nTable)  delete
tmp;
                tmp=tmp2;
     //         tmp=tmp->next;
   }
 }

 bool SoundTableIsPlaying(int nTable)
 {
   if(nTable<0) return false;
   //if(CurrentTableIndex<0) return false;
   TSoundThread*tmp=MT_FIRST;
   while(tmp!=NULL)
   {
          //   TSoundThread*tmp2=tmp->next;
                if(tmp->SoundDescriptor.table==nTable)  return
true;
                //tmp=tmp2;
                tmp=tmp->next;
   }
   return false;
 }

 void SoundTableWait(int nTable)
 {
   if(nTable<0) return;
   while(SoundTableIsPlaying(nTable))
   {
   __NOP();
   }
 }

 void AllSoundTableTerminate()
 {
   MTSoundsTerminatePlaying();
   CurrentTableIndex=-1;
 }

 float CalcSoundSpeed(uint16_t modulv,uint16_t kplayspeed)
 {
   //to do!
 return 1.0;
 }

 /*
 int8_t CheckCondition(TCondition condition,uint16_t value)
 {
   if (condition.Motoronly&&(!MotorOn) )return 1;
 if (modulv<condition.vmin) return -1;
 if (modulv>=condition.vmax)return 1;
 return 0;
 }
 */
```

```
 void SoundTableEffectIterator(uint16_t modulv)
 {
   TSoundTable  *  CurrentSTable=(TSoundTable  *)&SoundSheme-
>SoundTables[CurrentTableIndex];
   if((CurrentSTable==NULL)||(CurrentTableIndex<0)) return;
   TSoundThread*tmp=MT_FIRST;
   while(tmp!=NULL)
   {
   if (tmp->indexintable>=0)
     {
     TSoundTableEntry Sound=  CurrentSTable->Sound[       tmp-
>indexintable];
   //     TCondition condition=Sound.condition;
   //     tmp->enabled=    CheckCondition(condition,modulv);//
(modulv>=condition.vmin)&&(modulv<=condition.vmax);
     if (!tmp->enabled)tmp->Rewind();
      else
           tmp->speed=CalcSoundSpeed(modulv,Sound.playspeed);
     }
     tmp=tmp->next;
   }
 }
 bool SoundTableIterator(uint16_t modulv,int16_t a)
 {
     TSoundTable  *  CurrentSTable=(TSoundTable  *)&SoundSheme-
>SoundTables[CurrentTableIndex];
   if((CurrentSTable==NULL)||(CurrentTableIndex<0))     return
false;
 return false;
 }

 void DoChangeEffects(uint16_t modulv,int16_t a)
 {
 if (!SoundTableIterator(modulv,a))
     SoundTableEffectIterator(modulv);
 }


 void PlayTable_Event (uint8_t Event_ID) {
   for (int i=0;i< SoundSheme->soundTablePrimaryNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
PlayTable(i);

   for (int i=0;i< SoundSheme->soundTableSecondaryNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
PlaySecondaryTable(i);

   for (int i=0;i< SoundSheme->soundTableRandomNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
PlayRandomTable();

 }
```

```
 int PlaySingleTable_Event  (uint8_t Event_ID)
   {
   for (int i=0;i< SoundSheme->soundTablePrimaryNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
{PlayTable(i);return i;}

   for (int i=0;i< SoundSheme->soundTableSecondaryNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
{PlaySecondaryTable(i); return i;}

   for (int i=0;i< SoundSheme->soundTableRandomNumber;i++)
    if         (SoundSheme->SoundTables[i].Event_Id==Event_ID)
{PlayRandomTable();return i;}
   return -1;
 }
 void PlaySingleTable_Event_and_Wait  (uint8_t Event_ID)
 {
   int i=PlaySingleTable_Event(Event_ID);
   if (i<0
 ) return;
   SoundTableWait(i & 0xff);
 }
```