МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНА ТЕХНОЛОГІЯ АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАХИЩЕНОГО ЗВ'ЯЗКУ

Здобувач освіти гр. ІК.м–11 Владислав ЖУРАКУЛОВ

Науковий керівник,
к.ф.-м.н., ст. викладач Дмитро ВЕЛИКОДНИЙ

В.о. завідувача кафедри
к.т.н., доцент Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
«_____» _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Журакулову Владиславу Вячеславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Інформаційно-комунікаційна технологія забезпечення захищеного зв'язку*

затверджую наказом по СумДУ від «____» _____ 20 ___ р. № _____

2. Термін здачі здобувачем кваліфікаційної роботи_____

3. Вхідні дані до кваліфікаційної роботи_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки
1) Інформаційний огляд;
2) Вибір методу рішення;
3) Практична реалізація.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р. № _____

Науковий керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Інформаційний огляд		
2.	Вибір методу рішення		
3.	Практична реалізація.		

Здобувач вищої освіти _____
(підпис)

Науковий керівник _____
(підпис)

РЕФЕРАТ

Записка: 60 стор., 44 рис., 1 додаток, 15 літературних джерел.

Мета роботи — створення й програмна реалізація прототипу комунікатору з шифрованим зв'язком.

Об’єкт дослідження — методи, дослідження й розробка прототипу комунікатору з шифрованим зв'язком.

Методи дослідження — експеримент, наукове моделювання, прототипування.

Результати — Проведений аналіз різних інформаційних джерел, у тому числі технічної літератури, методів та інструментів. Вибрані методи рішення, основуючись на їх характеристиках. Розроблений й успішно протестований прототип комунікатору на основі плат Arduino й радіомодемів LoRa з шифруванням AES і ChaCha20.

Ключові слова: комунікатор, радіомодем, LoRa, Arduino, криптографія.

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНА ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ ЗАХИЩЕНОГО ЗВ'ЯЗКУ, INFORMATION AND COMMUNICATION TECHNOLOGY OF HARDWARE AND SOFTWARE OF SECURE COMMUNICATION

ЗМІСТ

ВСТУП

Інформація - це один з найцінніших ресурсів всіх часів й питання його транспортування, захисту й збору є найбільш актуальними в Цифровому Столітті. Озираючись навколо, кожен бачить пристрої які замінили нам органи й частини тіла, економлять наш час й сили. Так, наприклад, ми бачимо перед собою екран. Звичайний екран демонструє нам інформацію про навколишній світ і під впливом певних маніпуляцій людини ознайомлює її з різною інформацією.

Кожна людина аналізує себе й навколишній світ. Інформація є інструментом, деяка конфіденційна інформація може бути скопійована й використана проти її автора або у розріз з його баченням світу. Кожен наш крок в Інтернеті залишає відбитки, які використовуються в промислових масштабах проти волі й права їх творців. Саме тому, у час коли в більшості країн світу по словам й діям з'являється контекстна реклама, коли людей садять за грати через реакцію на пост, коли приватні розмови можна почути з екранів телевізорів - люди починають шукати захищені способи передачі інформації. Але нічого у цьому світі не може залишатися порожнім: якщо є попит - буде пропозиція.

Нічого масового, що може легко передати різні зашифровані дані на значні відстані без мережі Інтернет я не знайшов. Враховуючи різні фактори, постала необхідність у створенні пристрою, що дав би можливість передавати конфіденційну інформацію різних типів між людьми на значні відстані. Так з'явилася ідея комунікатору.

Фактором, що завдав значного впливу на розробку стала війна. Опускаючи всі труднощі у ці важкі часи, з'явився додатковий попит на комунікатори з шифруванням для військових. А саме, на передачу даних з датчиків тепла, вологи, GPS, радіації, тощо по шифрованим каналам зв'язку. Враховуючи необхідність, постало питання реалізації додаткових модулів в прототипах на основі плат Arduino для успішного виконання різних видів задач.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд предметної області

Для передачі інформації у просторі, одним з найефективніших способів є використання радіохвиль. Тут треба враховувати наступні фактори:

- залежно від частоти, треба створити певні умови для їх використання: на малих частотах (довгі та середні хвилі) треба великі антени, високі ж частоти мають низький радіус дії - відповідно потребують ретрансляторів для передачі інформації на значні відстані;
- більшість частот зарезервована, пристрої у продажу мають відповідні заглушки для уникнення їх використання. Тому прототип найліпше запускати за аматорською частотою 434 МГц або в спектрі 410-493 МГц;
- для збереження конфіденційності слід використати надійні та відкриті алгоритми шифрування, щоб уникнути криптографічних вразливостей. Слабкі шифри минулих сторіч, повтори даних, відсутність очищення даних після їх використання, тощо.

Вибравши середовище для передачі інформації, потрібно визначитися з основою для пристрою. Серед плат й контролерів, на яких можна зробити пристрій є досить велика кількість варіантів. Найпоширенішим й, що не менш важливо, доступними варіантами є плати: Arduino, BeagleBone, ESP8266, Teensy, Raspberry Pi, NodeMCU, STM32, Particle Photon, MSP430. Опишу кожну з них, щоб розуміти переваги розробки й вибору функціоналу.

Arduino

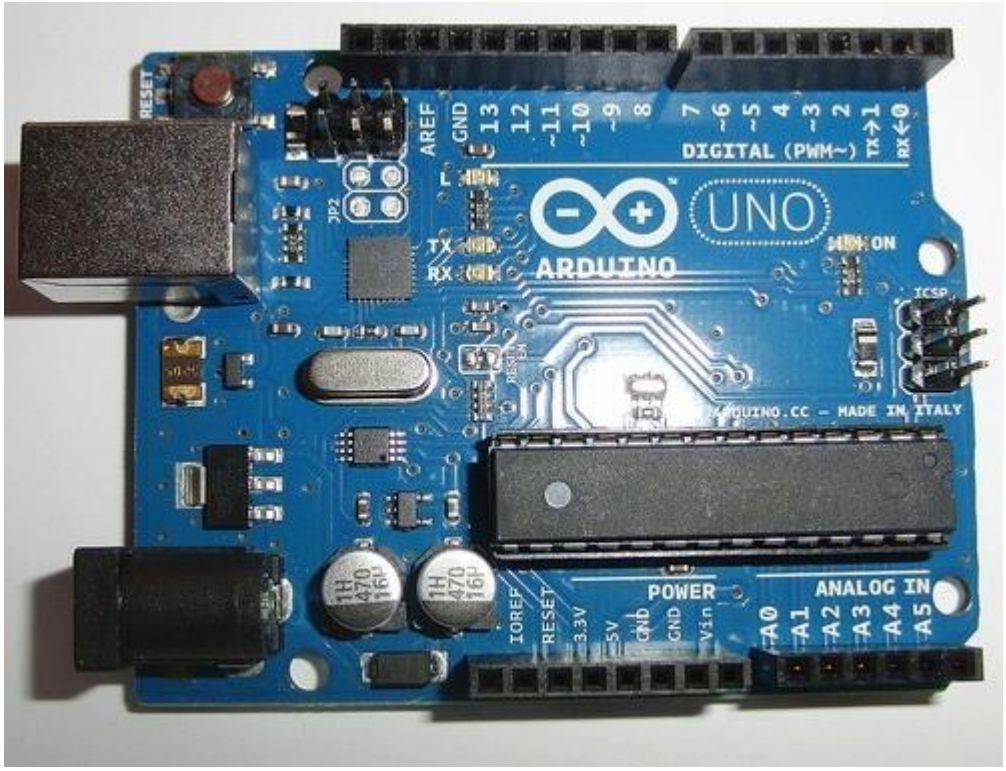


Рисунок 1.1 – Arduino UNO

Плата Arduino має безліч модифікацій і поширюється в різних моделях. Одна з найпопулярніших серед користувачів - Arduino Uno. Це найстаріший пристрій. Він з'явився у 2005 році, як інструмент для студентів. Потім пристрій було модифіковано, покращено і клоновано безліч разів. Мікрокомп'ютер Arduino зробив революцію в програмуванні та електроніці.

Переваги Arduino: простота розробки (не потрібно нічого налаштовувати, збирати повноцінну операційну систему Linux. Достатньо написати код, компілювати й завантажити на пристрій), низька ціна (офіційна версія Arduino Uno коштує 25$, але можна замовити клон за 5$), відмовостійкість (як показав час - остаточно перестає працювати після запаху палаючої пластмаси), розширюваність (може використовувати великий спектр модулів), й наявність знань з програмування даної з курсів в університеті і радіолюбителів.

Недоліком Arduino Uno є те, що тут використовується чип ATMEGA CPU, який має невелику кількість пам'яті та пристрій зберігання. Цю проблему можна нівелювати додатковою SD картою.

Raspberry Pi ZERO

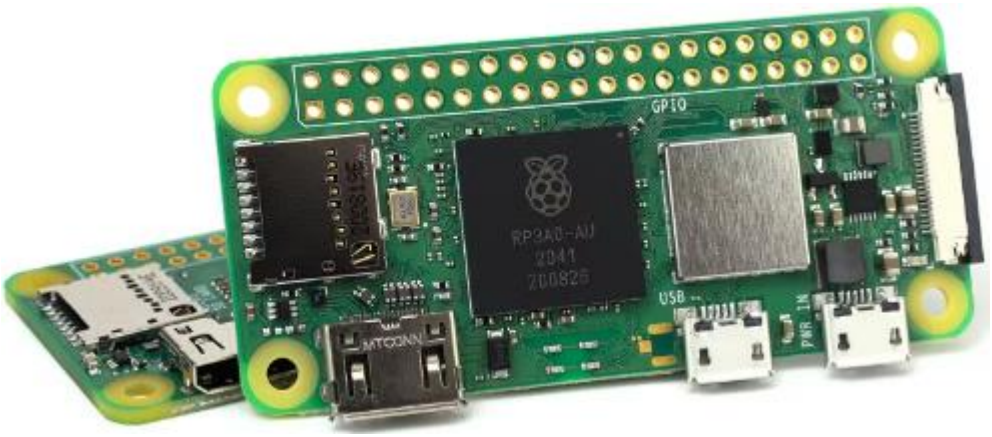


Рисунок 1.2 – Raspberry Pi ZERO

Ця плата може підтримувати деякі операційні системи, на кшталт Raspbian і Linux. Має вбудовані 2 порти microUSB, вбудований процесор із частотою 1 ГГц і 512 Мб оперативної пам'яті. Без Ethernet, Bluetooth або Wifi але має перехідники під розширення.

NODEMCU

NodeMCU - це мікроконтролер, який на відміну від Arduino використовує 8-бітний ATMEA з частотою 16 МГц, тут використовується чипсет ESP8266 з 32 бітовим процесором Tensilica Xtensa LX106 80 МГц, має Wifi, вбудована підтримка TCP/IP, 4 мегабайти вбудованого сховища й 20 кб ОЗП й 10 виходів GPIO.

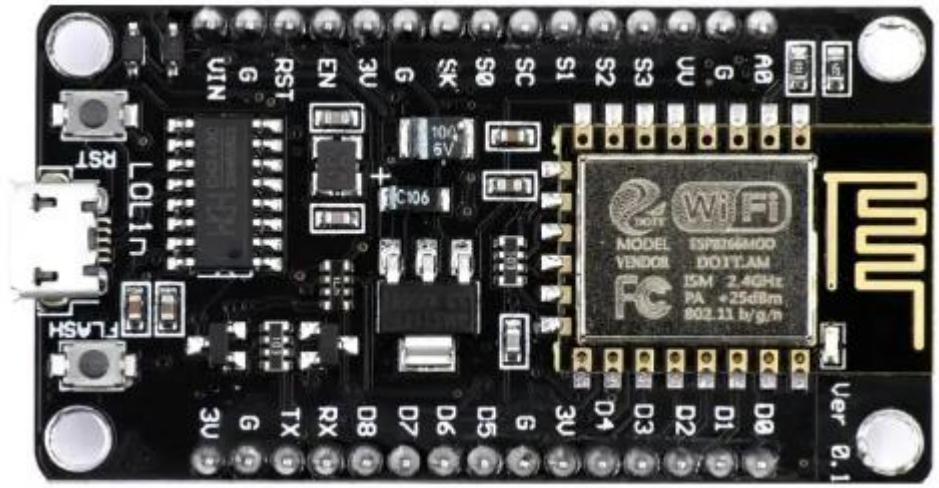


Рисунок 1.3 – NodeMCU

До пристрою можна підключати різні компоненти, як-от монітори, сенсори або сервоприводи. Як і Arduino, його дуже просто використовувати - достатньо написати код і завантажити його на пристрій через USB. Програми пишуться на Lua - це інтерпретована мова програмування, чи просто кажучи скриптований C++.

PARTICLE PHOTON

Particle Photon - це пристрій для реалізації різних веб-проектів. Пристрій постачається з Wifi. Як процесор використовується Cortex ARM M3 з частотою 120 МГц. Писати програми для нього потрібно так само, як і для Arduino.

Рисунок 1.4 – Particle Photon

Особливістю Particle Photon є те, що програмування використовує хмару - це дозволяє під'єднатися до пристрою за допомогою додатку на телефоні й прив'язати пристрій до акаунту. Як альтернатива є CLI (Command-line Interface) - рядок адміністратора для керування платою.

ESP8266

Рисунок 1.5 – ESP8266

ESP8266 (рис., 1.5) - це мікроконтролер із підтримкою Wi Fi, що може програмуватися в тій самій Arduino IDE. Але для його живлення потрібно подавати 3.3 вольта, а не 5. Пристрій постачається з вбудованим регулятором живлення та кількома портами вводу-виводу.

TEENSY

Рисунок 1.6 – TEENSY

Teensy - це компактна платформа для розробки й альтернатива Arduino, яка може використовуватися для створення будь-якого DIY (Do It Yourself - зроби сам) проекту. Тут є завантажувач, за допомогою якого можна завантажувати програму навіть з USB. Пристрій надає можливість емулювати будь-який USB-пристрій, а як процесор використовується ARM Cortex M4 з частотою 180 МГц, і 256 Кб оперативної пам'яті. Підтримує Arduino IDE.

BEAGLEBONE

Рисунок 1.7 – BEAGLEBONE

Пристрій використовує ARM Cortex V8 з частотою 700 МГц і 256 мегабайт оперативної пам'яті DDR2, а також флешку об'ємом 4 Гб. Як мови програмування

можуть використовуватися Python, C, C++, PHP, JavaScript. Пристрій підтримує установку SD карти, а також є USB порт, через який можна підключати різні розширення, наприклад, Ethernet або інший комп'ютер.

MSP430

Рисунок 1.8 – BEAGLEBONE

MSP430 - це альтернатива Arduino, дуже схожа на оригінальну плату, але споживає дуже мало енергії, завдяки використанню 16-бітного MCU. Як середовище для розробки програм може використовуватися Energia IDE. Мікроконтролер має власну архітектуру, що і виділяє його серед інших.

STM32

Рисунок 1.9 – STM32

STM32 (рис., 1.9) - це дешевий 32-бітний мікроконтролер, від STMicroelectronics. Тут використовується своє середовище розробки Keli Linux, а також програматор ST-Link. Пристрій використовує чип ARM Cortex 32-bit M3

з тактовою частотою 24 МГц і 8 кб оперативної пам'яті. Серед інших переваг можна відзначити низьке споживання енергії й обробку цифрових сигналів.

1.2 Аналіз аналогів

Як таких аналогів багатофункціональному комунікатору захищеного зв'язку на апаратній базі LoRa та Arduino немає але є багато проектів по всьому світу з аналогічною технологією, переважно зі слабким захистом інформації. Рації, пейджери чи телефони - це зовсім інші пристрої, тому не можна вважати їх аналогами пристроїв за технології LoRa, через принципово різні способи реалізації з'єднання. Через те, логічно проаналізувати у першу чергу технологічні рішення на основі технології LoRa.

Освітлення Бостону

Рисунок 1.10 – Ліхтарі в Бостоні

Приклад Бостону (рис., 1.10) в Массачусетсі у США показує, що можливо з LoRa. У місті більш ніж 67,000 вуличних ліхтарів. Вони освітлюють жваві вулиці, історичні пам'ятки й площі. З понад 1,600 розподільчих коробок і 32,000 кришок люків, підтримання цієї інфраструктури вимагає значних загальних зусиль - особливо коли лампи інтенсивно використовуються в зимові місяці.

Освітлення Талліну

Встановлення LoRa протестовано й на всіх лампах у порту, якими можна дистанційно керувати через мережу LoRaWAN. Контролер лампи підключений до шлюзу LoRaWAN через радіосистему, що належить порту. Інші шлюзи поблизу також підтримують це з'єднання. Платформа безпечного концентратора Інтернету речей візуалізує зашифровані дані з кінцевих вузлів через серверну частину LoRaWAN.

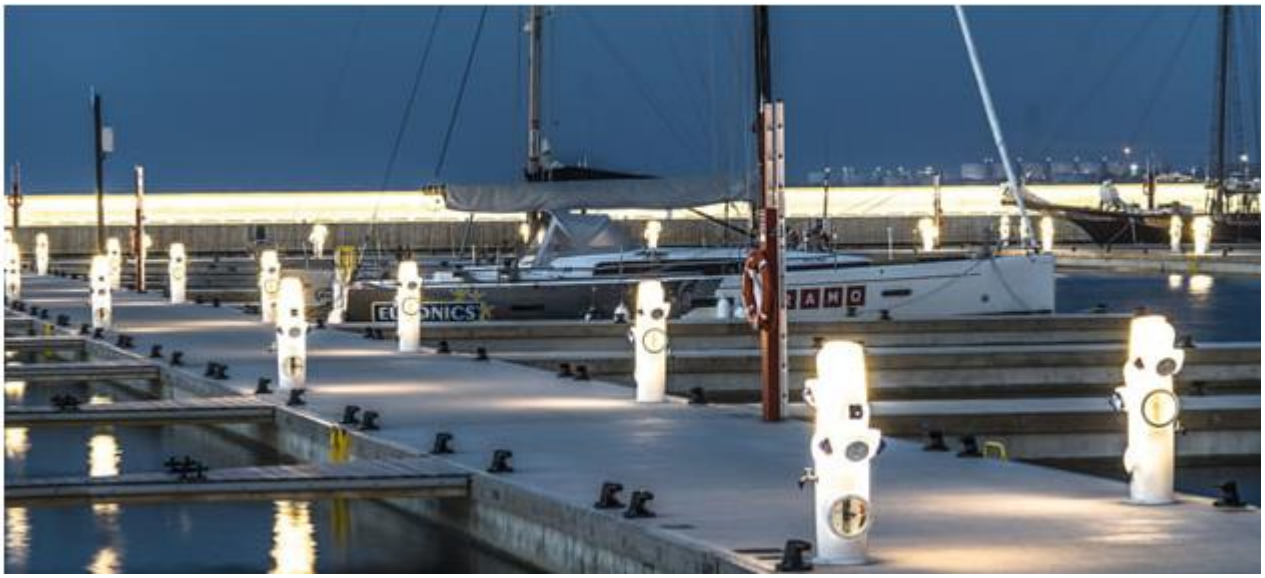


Рисунок 1.11 – Портове освітлення біля Талліна на півострові Какумяе (Естонія).

На рисунку 1.11 можна побачити портове освітлення біля Талліна на півострові Какумяе (Естонія).

Використання LoRa у Нідерландах

В Нідерландах оператор KPN (Koninklijke KPN N.V) організовує мережі по всій країні для будь-яких проєктів подібного типу. Мережі LoRa використовують порти Ротердама та вантажоперевізники для відстежування контейнерів й товарів по всій країні, утилізації відходів та управління каналами води.

Рисунок 1.12 – Канал в Нідерландах.

На рисунку 1.12 можна побачити канали й іншу інфраструктуру, що використовує LoRa модеми.

1.3 Постановка задачі

Метою кваліфікаційної магістерської роботи є створення й програмування прототипу багатофункціонального комунікатору захищеного зв'язку на апаратній базі LoRa та Arduino й можливістю подальшої модифікації.

Реалізація поставленої мети визначає наступні задачі:

- зібрати прототип комунікатору;
- запрограмувати комунікатор;
- протестувати комунікатор.

Прототип може бути розроблений в декількох екземплярах для подальшого тестування. Програмна частина має працювати без критичних помилок. Тестування прототипу, в першу чергу, має перевірити основні можливості комунікатору, що потребують найбільшої уваги й допомогти локалізувати й вирішити критичні баги програмного забезпечення комунікатору. Модулі можуть бути реалізовані й протестовані в залежності від їх актуальності.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Опис модулю LoRa

LoRa - це модуляція цифрового зв'язку з великим радіусом дії (буквально перекладається як Long Range). Не потребує великих антен, чи значних кількостей енергії для роботи. Гарним вибором є модуль LoRa E22-400M30S - він базується на модулі SX1268 й підсилювачах потужності і прийому (PA: (power amp) amplifies when transmitting, LNA: (low noise amp) amplifies when receiving). Детальний опис модулів від фірм розробників й всього функціоналу я знайшов в наступних мануалах: E22-400M30S User Manual [1]; SX1268 Long Range, Low Power, sub-GHz RF Transceiver User Manual [2].

2.2 Опис плати Arduino

Arduino Mega 2560 - це пристрій на основі мікроконтролера ATmega2560 (datasheet). До його складу входить усе необхідне для зручної роботи з мікроконтролером: 54 цифрові входи/виходи (з яких 15 можна використовувати як ШІМ-виходи), 16 аналогових входів (які також можна використовувати як цифрові), 4 UART (апаратні приймачі для реалізації послідовних інтерфейсів), кварцовий резонатор на 16 МГц, роз'єм USB, роз'єм живлення, роз'єм ICSP для внутрішньосхемного програмування і кнопка скидання. Для початку роботи з пристроєм досить просто подати живлення від AC/DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою USB-кабелю. Arduino Mega сумісний з більшістю плат розширення, розроблених для Arduino Duemilanove і Diecimila.

На платі Mega 2560 версії R2 додано резистор, що підтягує до землі лінію HWB мікроконтролера 8U2. Такий захід дає змогу спростити процес оновлення прошивки та перехід пристрою в режим DFU. Вся документація й ArduinoIDE

доступні на офіційному сайті [3]. Екрани для плати Arduino були підібрані відповідно до вимог, технічна література зазначена на їх офіційних сайтах [4], [5].

2.3 Опис Функціоналу

Комунікатор може передавати різні дані (текст, погодні чи дозиметричні дані, тощо) в шифрованому вигляді, виводити на екран інформацію й реалізувати взаємодію з користувачем через сенсори на ньому. Екран має вбудовану microSD карту, що значно підвищує можливість запису софту й даних користувача. Комунікатор, залежно від необхідності, оснащується різними модулями: GPS, датчики температури або вологи, тощо. Візуальна частина зроблена в системному ПЗ для прикладних програм Labview. Частина коду надрукована вручну, частина сґенерована Labview через блок схеми. Приклад роботи можна побачити на рисунках 2.1, 2. 2.

Рисунок 2.1 – Екран налаштувань для передачі даних.

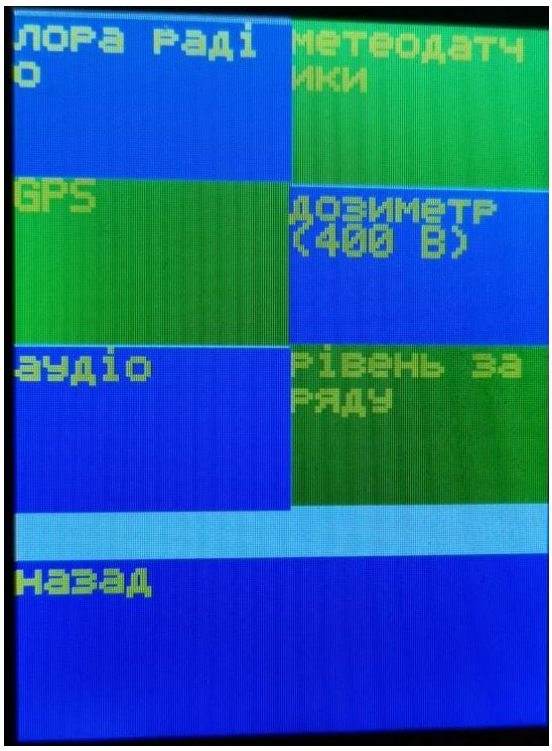


Рисунок 2.2 – Екран вибору тестових підпрограм для кожного модулю.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка прототипу

Розробку прототипу почнемо з її основ - плати комунікатору й апаратної платформи Arduino Mega, після чого поетапно розберемо всі приєднані до них компоненти.

Друкована плата

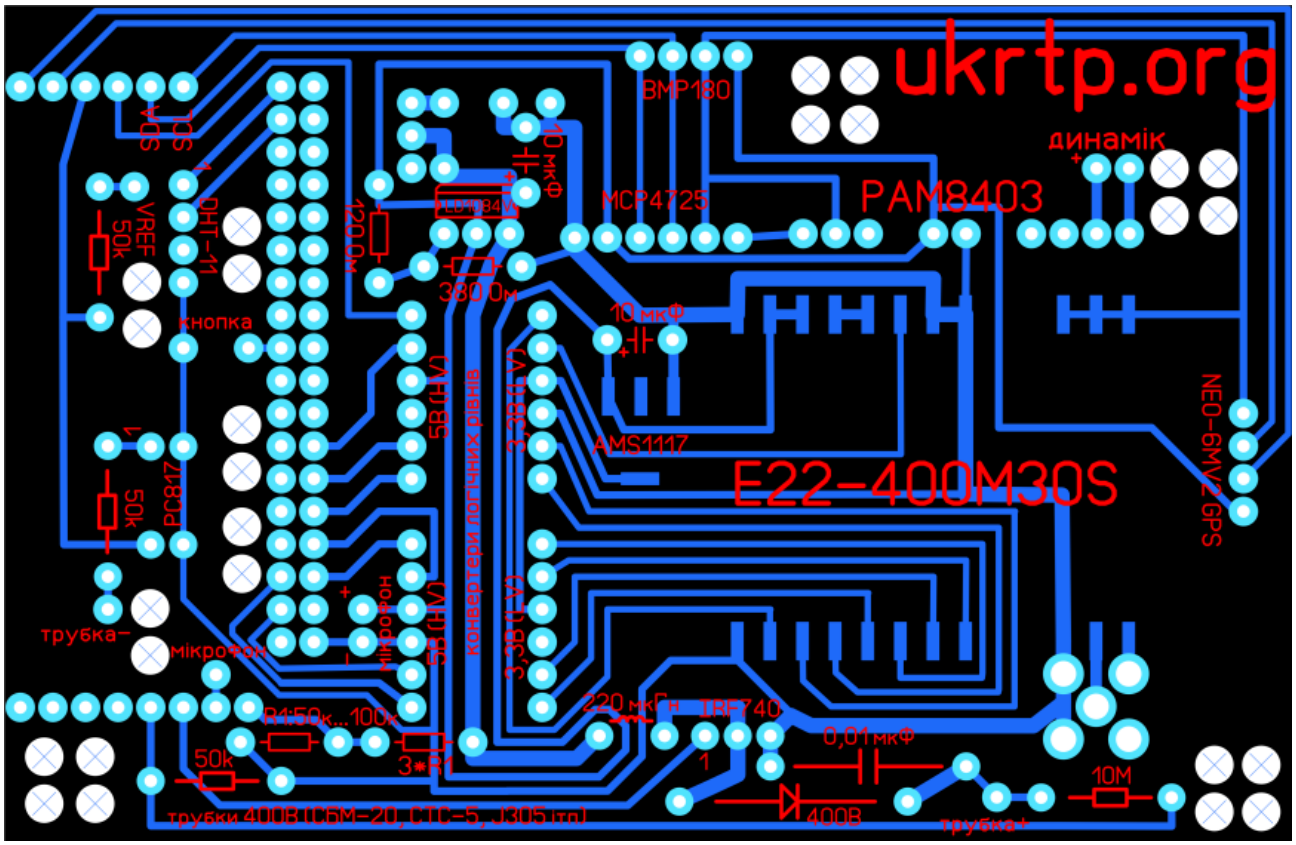


Рисунок 3.1 – Схема друкованої плати.

Так виглядає кінцевий варіант друкованої плати (рис., 3.1). Розробка структури проходила в програмі Sprint-Layout для подальшого друкування під замовлення. Звісно були і ранні прототипи, на яких відпрацьовувалася робота компонентів комунікатору. Варіанти попередніх прототипів були досить варіативними й різної якості, тому зупинюся на останньому. На останньому прототипі доріжки, прорізи в текстоліті й інші елементи виповненні якісно на

заводському рівні по схемі зазначеної на рисунку 3.1.

Друкована плата спеціально розроблена так, щоб до неї можна було додати модулі. Основні модулі підписані прямо на схемі, що легко побачити на рисунку 3.2. Друкована плата під'єднується до апаратної платформи Arduino відповідними міжплатними з'єднаннями, котрі можна побачити на рисунку 3.3.

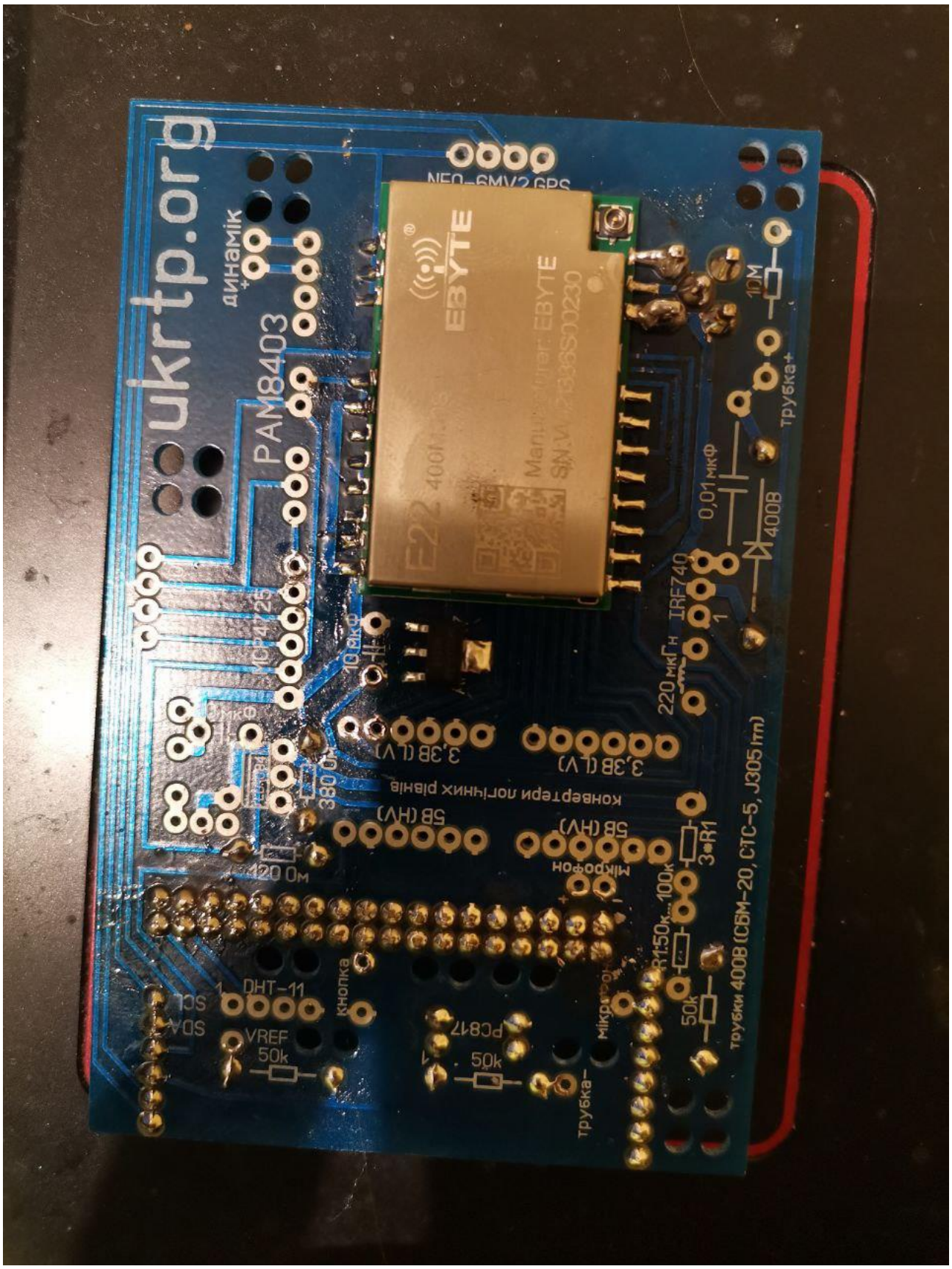


Рисунок 3.2 – Друкована плата з LoRa (верх).

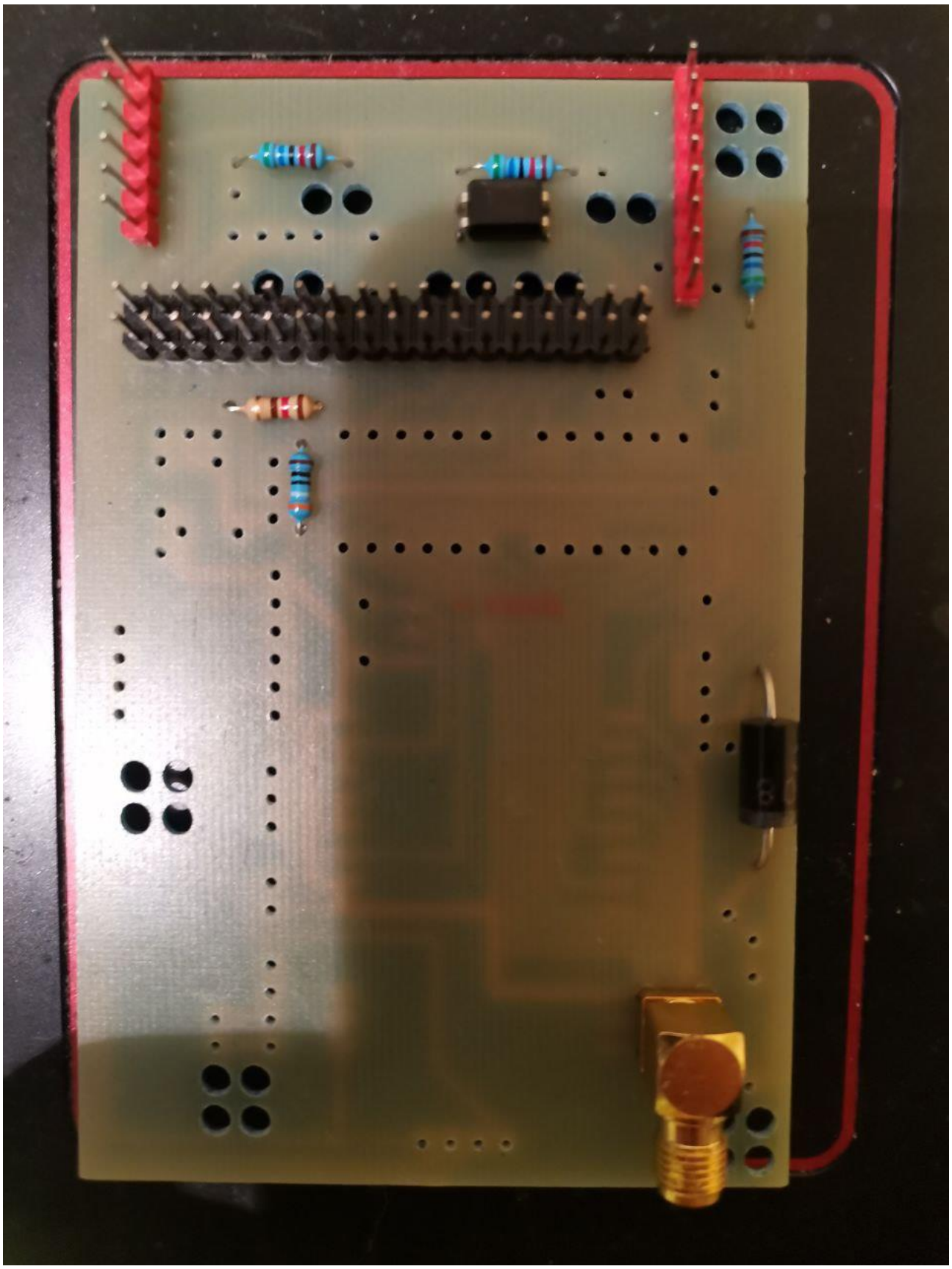


Рисунок 3.3 – Друкована плата з LoRa (низ).

Основні частини комунікатору:

- апаратна платформа Arduino Mega;
- друкована плата;
- LoRa радіомодем;
- екран з microSD карткою;
- акумуляторний відсік;
- аудіо (мікрофон, ЦАП - цифро-аналоговий перетворювач, регульований аудіо підсилювач, динамік).

Arduino Mega

На рисунку 3.4 можна побачити підключену Arduino Mega 2560. Як бачимо, до платформи можна підключити різні датчики й модулі. На фото через breadboard підключені датчики вологи й GPS. У різних комбінаціях всі варіації пристрою були протестовані на breadboard з різними модулями й апаратними платформами (на рисунку 3.4 breadboard має не підключену Arduino Nano й датчик температури, які також використовувались для тестування можливостей комунікатору).

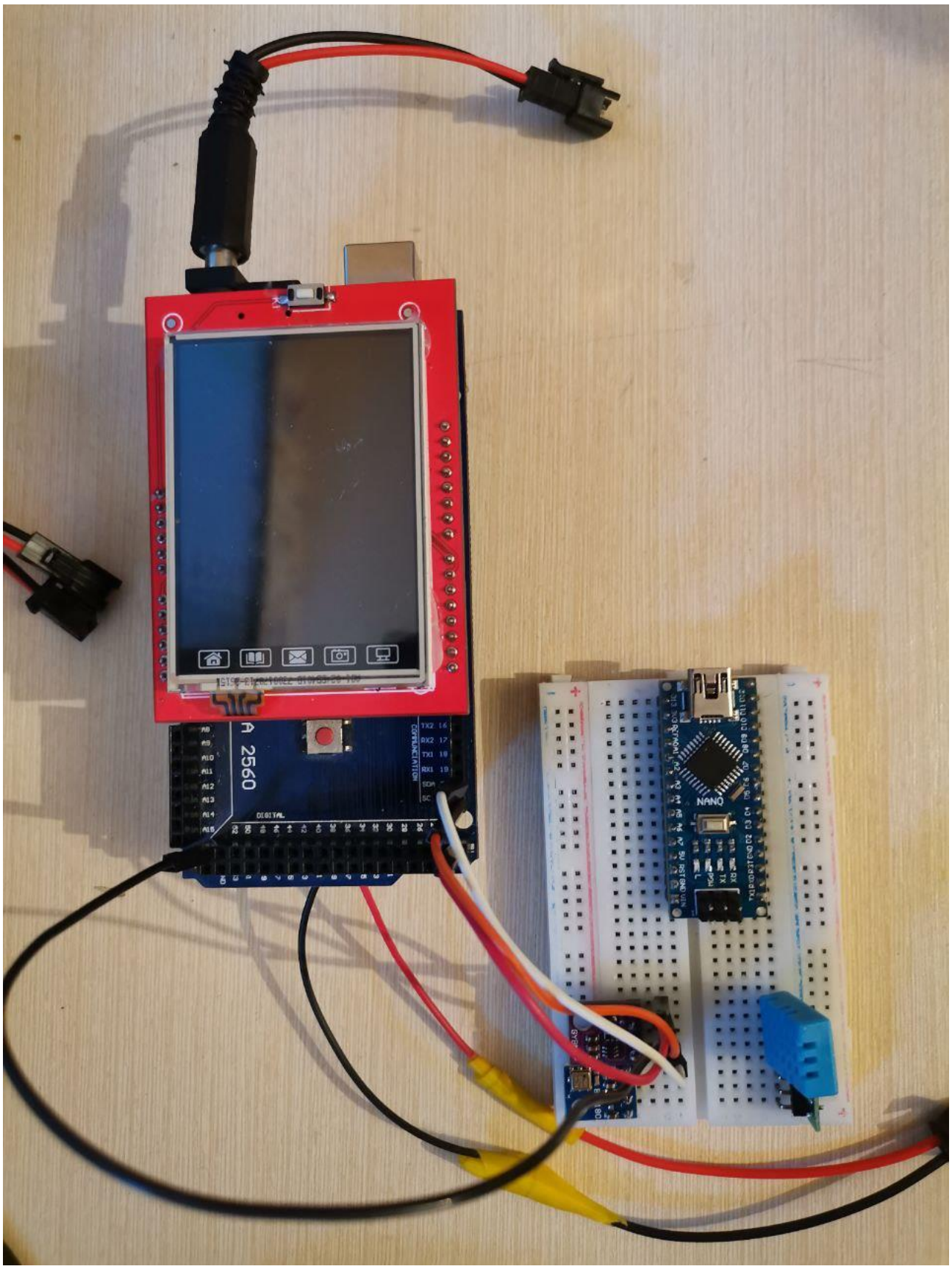


Рисунок 3.4 – Arduino Mega з екраном, GPS й датчиком вологи.

LoRa

На рисунку 3.5 можна побачити вибраний модуль LoRa, він був під'єднаний до Arduino через конвертори логічних рівнів. Це зроблено через те, що комунікаційні піни модуля LoRa працюють на 3.3 V, у той час як плата працює на 5V (живлення 5,2 V). Про різні технічні дані й тонкощі підключення радіомодему LoRa я дізнався з технічної літератури й статей присвячених даному радіомодему [6], [7], [8].

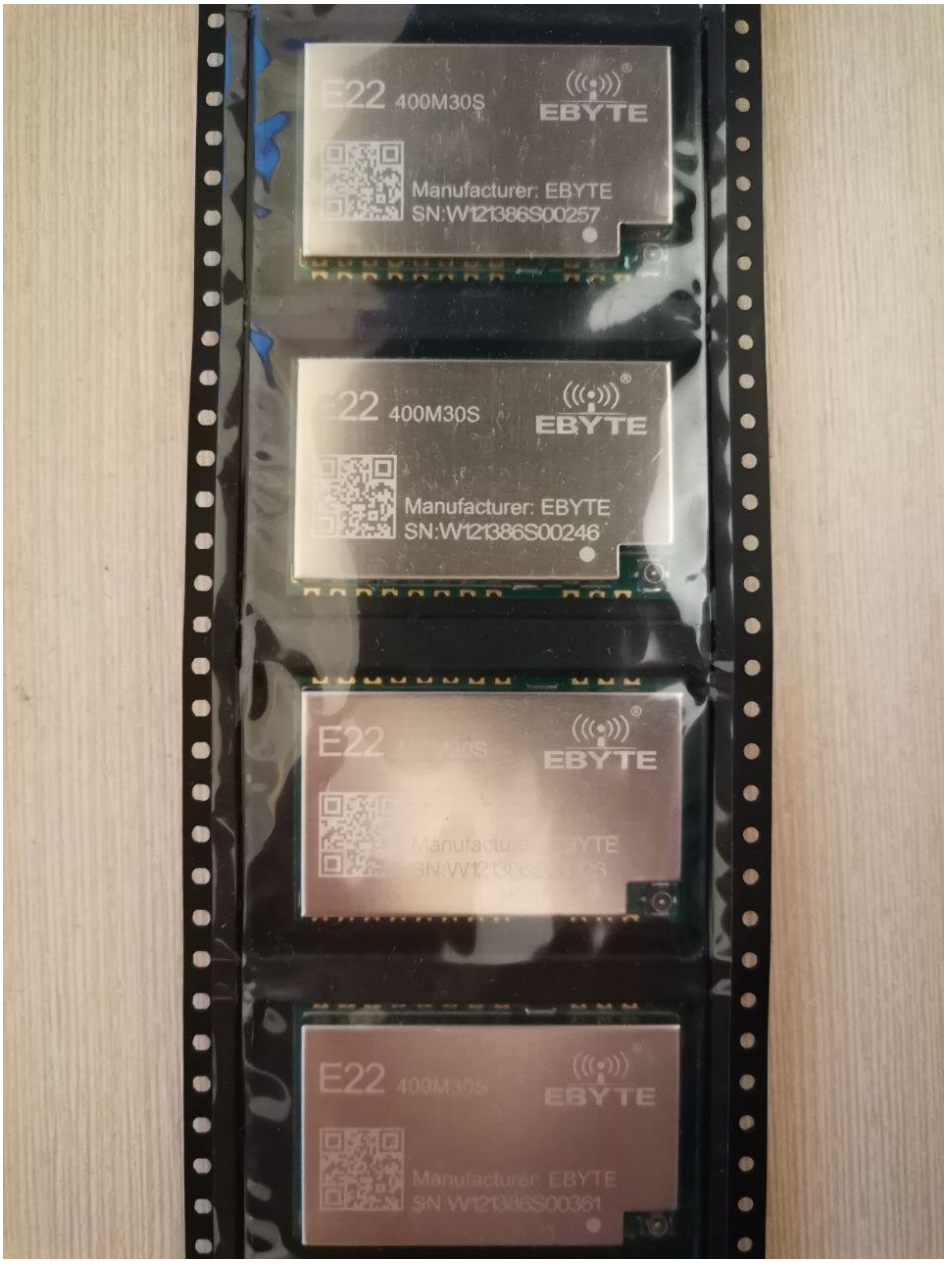


Рисунок 3.5 – Запаковані LoRa.

LoRa взаємодіє з Arduino за допомогою шини SPI (Serial Peripheral Interface) і кількох додаткових пінів. Вони мають наступний вигляд - рисунок 3.6

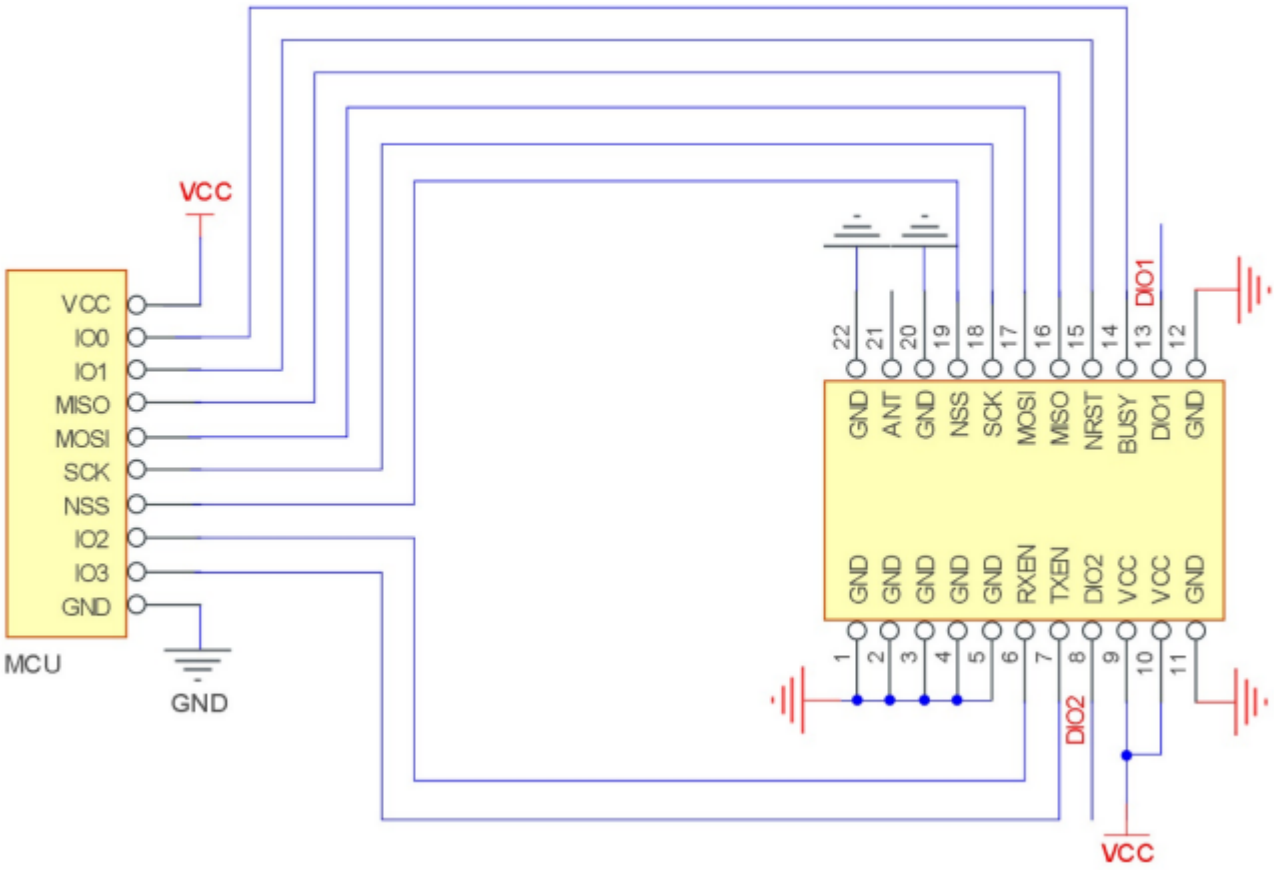


Рисунок 3.6 – Схема пінів для LoRa E22-400M30S [1].

- 1-5, 11-12 20, 22 піни це живлення мінус;
- 6 пін RXEN (Receive Enable) - дозвіл на прийом (активація підсилювача прийому, підключений до Arduino);
- 7 пін TXEN (Transmit Enable) - активація підсилювача на передачу (автоматично активується через сусідній пін DIO2);
- 8 пін DIO2 (Digital Input Output 2) може бути налаштований на те, щоб видавати логічне 1 при передачі даних (в схемі комунікатору він автоматично вмикає підсилювач потужності сигналу на передачу через з'єднання з 7-м піном);
- 9-10 піни це живлення + (підключені до стабілізатору напруги й видають 5.2 V);

- 13 пін DIO1 (Digital Input Output 1) використовується для отримання інформації радіомодему. Через нього отримується інформація про прийом пакету, в режимі передачі отримується інформація про закінчення передачі;
- 14 пін BUSY використовується для отримання інформації про стан модему, сигналізує коли модем зайнятий;
- 15 пін NRST використовується для перезавантаження модему;
- 16 пін MISO (Master Input Slave Output) передає дані від підпорядкованих пінів, до головних - з LoRa до Arduino. Один з SPI (Serial Peripheral Interface) портів;
- 17 пін MOSI (Master Output Slave Input) передає дані від головного, до підпорядкованих;
- 18 пін SCK задає від Arduino тактову частоту (один тік - один біт по всіх пінах), SPI порт;
- 19 пін NSS активує пристрій на шині SPI;
- 21 пін це антена (антену розташуємо найближче до антенного піна, щоб уникнути втрат енергії на передачу. Високочастотні сигнали мають властивість втрачати потужність на довгих доріжках).

LoRa модем був припаяний як велика SMD (Surface Mounted Device) деталь до контактних майданчиків (рис., 3.7).

Рисунок 3.7 – Запаяна LoRa й антена біля неї.

Екран з microSD картою

У більшості прототипів використаний екран ELECROW 2.8 TFT Touch Shield v4.3, даний екран встановлений до платформи Arduino. У шилд, до слоту для microSD встановлена карта пам'яті.

Рисунок 3.8 – ELECROW 2.8 TFT Touch Shield v4.3 [5].

На рисунку 3.8 екран ELECROW 2.8 TFT Touch Shield v4.3 240x320 пікселів використовує драйвер ILI9341 (для сенсорних екранів) й SPI (Serial Peripheral Interface) інтерфейс.

Акумулятор

В якості джерела енергії використовуються два акумулятори 18650 з'єднаних послідовно (максимальна вихідна напруга 8.4 V, зображені на рис., 3.9)

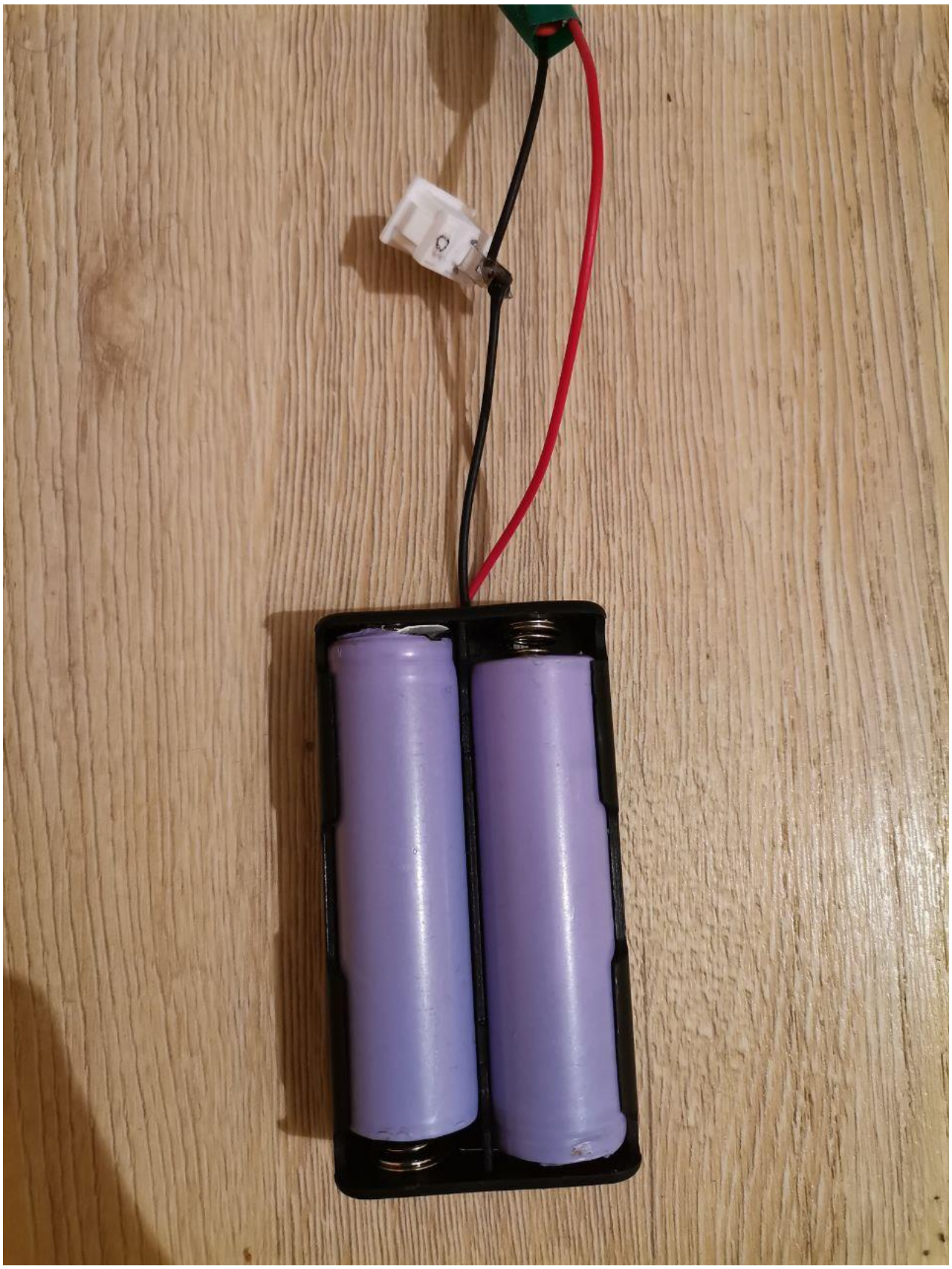
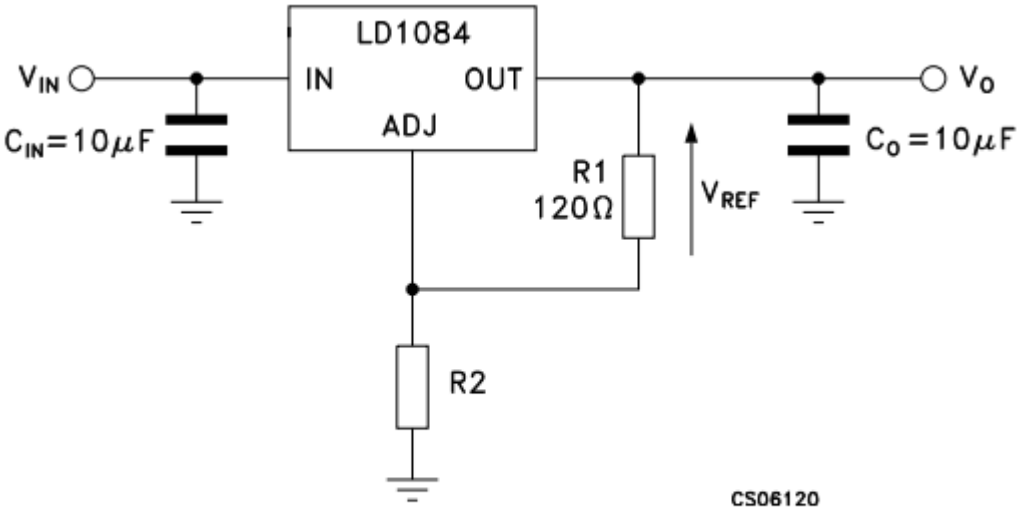


Рисунок 3.9 – Послідовно з'єднані акумулятори.

Через вимикач живлення подається на плату, де під'єднується до стабілізатору LD1084. Даний стабілізатор через два резистори налаштований на видачу 5.2 V (Напруга вище 5 V задля вищої потужності передавача).



$$V_O = V_{REF} \cdot \left(1 + \frac{R_2}{R_1}\right)$$

Рисунок 3.10 – Послідовно з'єднані акумулятори [2].

Після стабілізатора, 5.2 V йдуть на живлення модулю LoRa, ЦАП (цифро-аналоговий перетворювач аудіо), аудіопідсилювач, датчик тиску, конвертори логічних рівнів й GPS.

3.2 Програмна реалізація

Перш ніж почати опис програмної реалізації, зазначу, що надати весь код й прокоментувати всі функції в повному об'ємі я фізично не можу в межах магістерської роботи. Основна програма займає близько 13'900 рядків коду, більш того використовує багато бібліотек й іншого відкритого ПЗ (модифікована бібліотека шифрування ChaCha20 вказана в ДОДАТКУ), що вказано в даній роботі в повному обсязі.

Програмування плати Arduino здійснюється через стандартний Arduino IDE. Код скетчу генерується напівавтоматично саморобною програмою-кодогенератором АрдПрог зробленою на LabVIEW.

Arduino IDE, для коректної компіляції скетчу, потребує дві функції: "Void Loop" й "Void Setup".

Скетч генерований в АрдПрогу (рис., 3.11, 3.12, 3.13) являє собою машину станів (state machine), де кожен стан є окремою підпрограмою графічного інтерфейсу. Автоматично генерується код, що відповідає за роботу з сенсорним екраном. Кожна підпрограма складається з коду, що виконується на початку програми (задається вручну) з повторенням, і циклічним кодом (задається вручну й повторюється до переходу до іншої підпрограми). До кожної кнопки на екрані комунікатору може бути приєднаний шматок коду, який виконується в разі натискання на неї. Код відповідальний за умову спрацювання кнопки, тобто натискання на область кнопки на екрані генерується автоматично. Автоматично генерується код відображення інформації індикаторів (цифровий, булеан). Індикатори можуть бути задані вручну. Колір, розмір, назву й інші параметри кнопок можна задати в програмі АрдПрог. Текст кодується в однобайтовому кодуванні Windows 1251 через те, що однобайтове кодування значно менше займає пам'яті при записі кириличних символів (які в UTF-8 займають 2 байти). Це важливо для оптимізації роботи комунікатору. По мірі запису букви записуються в масив kbd (keyboard). По натисканні кнопки ОК стан прирівнюється до завчасно заданого значення (змінна klava_return_state), для повернення проводиться перехід до стану 73.

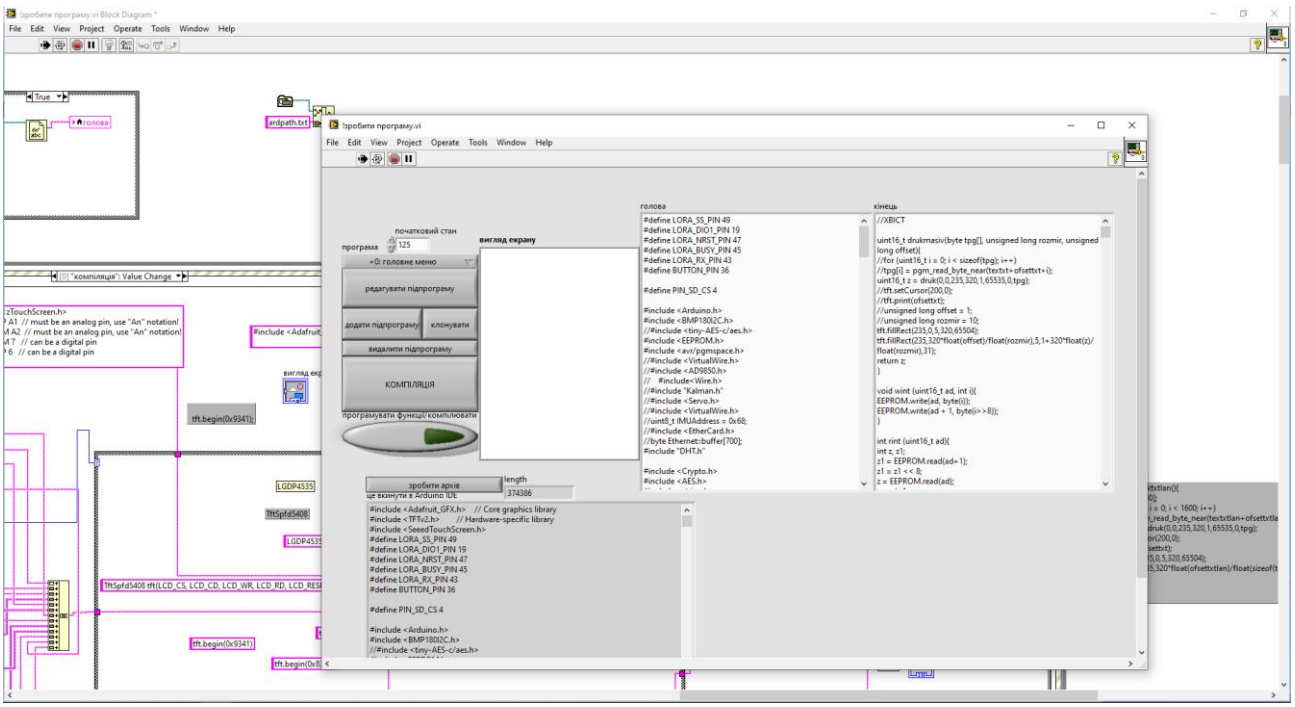


Рисунок 3.11 – Головне вікно програми АрдПрог.

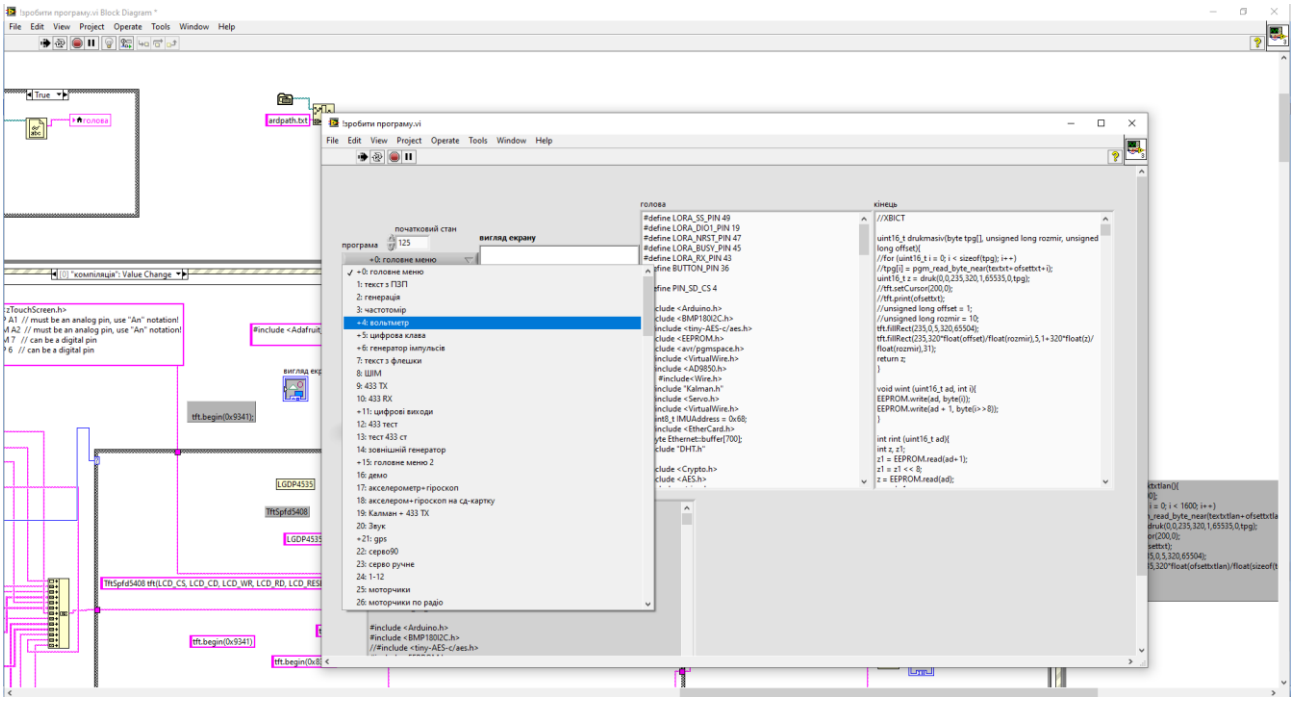


Рисунок 3.12 – Вибір підпрограм комунікатору.

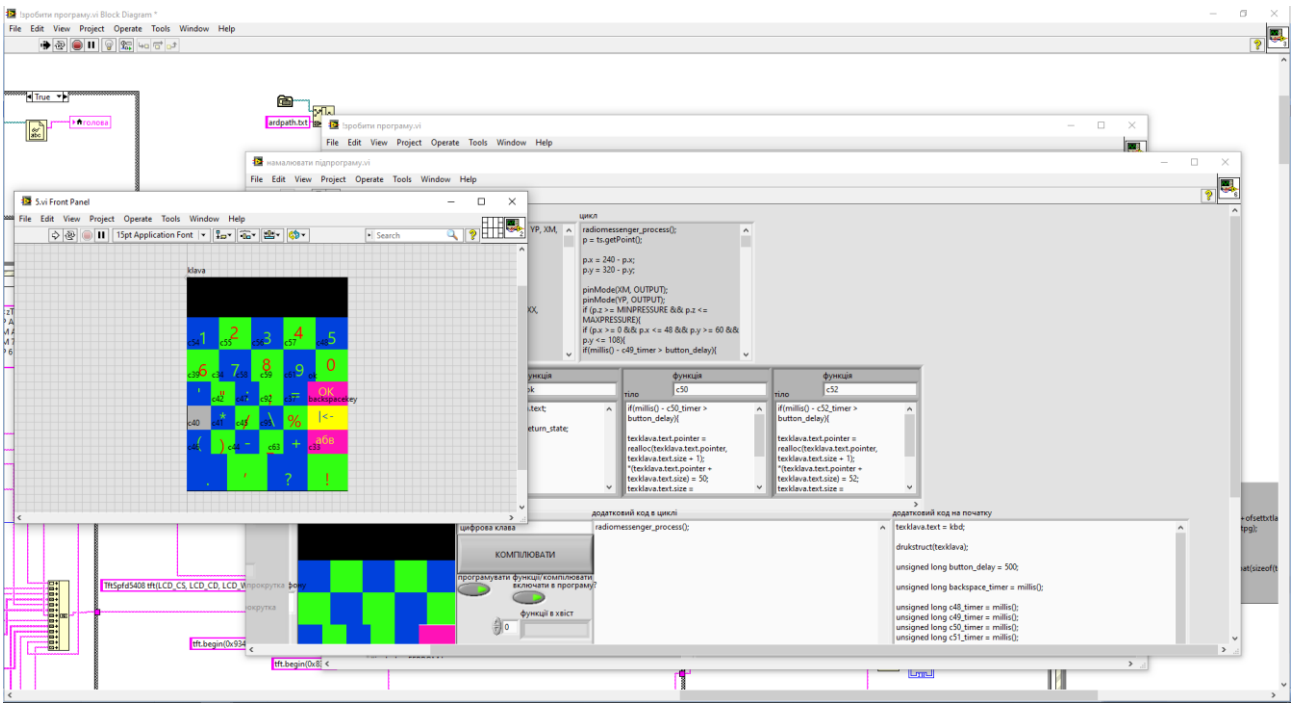


Рисунок 3.13 – Підпрограма цифрової екранної клавіатури комунікатору.

Масив байтів і функції роботи з ним

Перейдемо до клавіатури - за введення тексту відповідає група клавіатурних підпрограм.

За текст в прошивці радіокомунікатора відповідають структури (масиву байтів) типу struct byte_array (рис., 3.14).

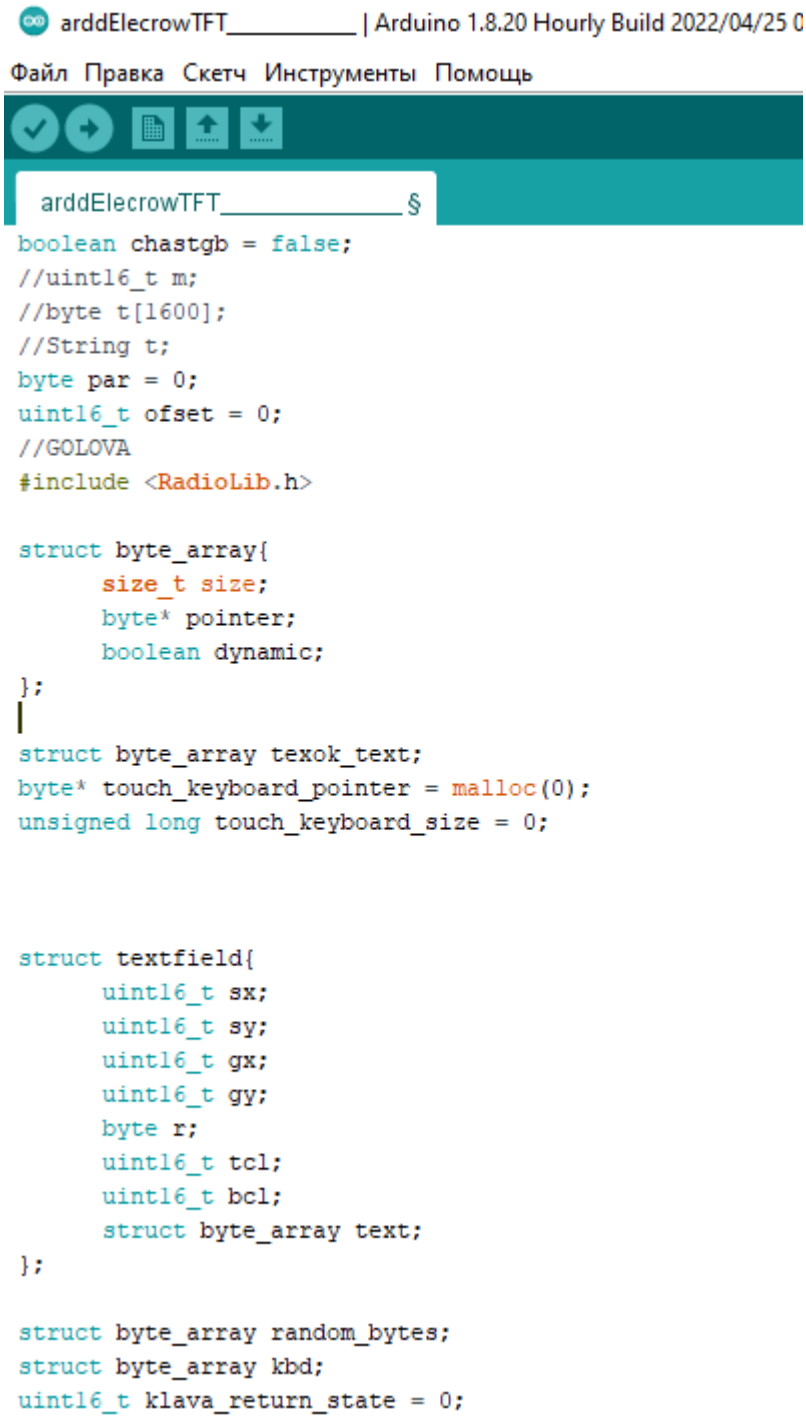


Рисунок 3.14 – Підпрограма цифрової екранної клавіатури комунікатору.

Опис елементів структури:

size - це розмір масиву.

pointer - вказівника на перший елемент.

dynamic - динамічний масив.

struct byte_array - ця конструкція взята з мови С без елементів ООП в С++. Це дає змогу працювати з полями цієї структури в будь-якому місці коду, що значно спрощує розробку в порівнянні з використанням об'єктів. Виключається необхідність писати методи для кожного одиничного випадку, у той час як для об'єктів треба писати методи до кожного окремого випадку роботи з їхніми полями й атрибутами.

byte_array_create - створює масив заданої довжини через масив байтів (текст) через malloc (рис., 3.15).

```
struct byte_array byte_array_create(size_t array_size){
struct byte_array returna;
returna.dynamic = true;
returna.size = array_size;
returna.pointer = malloc(array_size);
return returna;
}
```

Рисунок 3.15 – Функція byte_array_create.

byte_array_free - звільняє пам'ять масиву через free, якщо він динамічний (як приклад створений через byte_array_create) На рисунку 3.16 зображена функція byte_array_free.

```
void byte_array_free(struct byte_array array_to_free){
  if (array_to_free.dynamic){
    free(array_to_free.pointer);
  }
}
```

Рисунок 3.16 – Функція byte_array_free.

byte_array_replace_element - це функція (рис 3.17), що замінює елемент масиву. Заміна відбувається, якщо індекс елементу масиву лежить в межах розміру масиву; тоді функція повертає логічне так (true). Це на випадок випадкового виходу за межі виділеної під масив області пам'яті.

```
boolean byte_array_replace_element(struct byte_array array_for_replacement, size_t index_of_replace, byte element_to_replace){
  if (index_of_replace < array_for_replacement.size){
    *(array_for_replacement.pointer + index_of_replace) = element_to_replace;
    return true;
  }else{
    return false;
  }
}
```

Рисунок 3.17 – Функція byte_array_replace_element.

print_byte_array_bytes - функція для друку байтів з масиву в послідовний порт (Serial), що використовується для дебагу (рис., 3.18).

```
void print_byte_array_bytes (byte_array array_to_print){
  //Serial.write(array_to_print.pointer,array_to_print.size);


  Serial.println(array_to_print.size);
  for(size_t i = 0; i<array_to_print.size;i++){
  Serial.println(*(array_to_print.pointer + i));
  }
```

Рисунок 3.18 – Функція print_byte_array_bytes.

byte_array_get_element - дана функція (рис., 3.19) витягує заданий елемент масиву за його індексом (index_of_get). Функція також має запобіжник від виходу за межі масиву.

```
byte byte_array_get_element(struct byte_array array_for_get, size_t index_of_get){
  if (index_of_get < array_for_get.size){
    return *(array_for_get.pointer + index_of_get);
  }else{
    return 0;
  }
}
```

Рисунок 3.19 – Функція byte_array_get_element.

byte_array_subset - функція (рис., 3.20) витягує підмножину масиву за заданим індексом й довжиною. Якщо початковий масив динамічний, то копіювання здійснюється функцією memcpy. Інакше (якщо масив статичний - прописаний в постійній пам'яті Arduino) витягується відповідною функцією pgm_read_ptr, що відповідає за читання байтів з постійної пам'яті прошивки.

```
struct byte_array byte_array_subset(struct byte_array source_array,size_t index_of_subset,size_t size_of_subset){

  struct byte_array returna = byte_array_create(size_of_subset);

  if (source_array.dynamic){
    memcpy(returna.pointer,source_array.pointer + index_of_subset,size_of_subset);
  }
  else{
    for (size_t i = 0; i<size_of_subset;i++){
      *(returna.pointer + i) = pgm_read_ptr(source_array.pointer + i +index_of_subset);
    }
  }
  return returna;
}
```

Рисунок 3.20 – Функція byte_array_subset.

LoRa_settings_to_byte_array - функція (рис., 3.21, 3.22) бере структуру налаштувань LoRa (LoRa_settings) й перетворює на масив байтів (byte_array). Однобайтові поля структури записуються у відповідні елементи масиву байтів. Двобайтові числа в полях структури записуються наступним чином: молодший байт записується як є, а для запуску старшого число зсувається на 8 біт. Перед конверсією частота й ширина смуги множаться на 10, тому що є дробовими числами з роздільною здатністю 0.1 з отриманням цілого числа.

```
struct byte_array lora_settings_to_byte_array(struct lora_settings input){
  struct byte_array not_idiot_name = byte_array_create(10);

  uint16_t freq_int = input.freq * 10;
  uint16_t bw_int = input.bw * 10;
  byte_array_replace_element(not_idiot_name, 0, freq_int>>8);
  byte_array_replace_element(not_idiot_name, 1, freq_int);
  byte_array_replace_element(not_idiot_name, 2, input.power);
  byte_array_replace_element(not_idiot_name, 3, bw_int>>8);
  byte_array_replace_element(not_idiot_name, 4, bw_int);
  byte_array_replace_element(not_idiot_name, 5, input.cr);
  byte_array_replace_element(not_idiot_name, 6, input.preambleLength>>8);
  byte_array_replace_element(not_idiot_name, 7, input.preambleLength);
  byte_array_replace_element(not_idiot_name, 8, input.sf);
  byte_array_replace_element(not_idiot_name, 9, input.syncWord);

  return not_idiot_name;

}
```

Рисунок 3.21 – Функція LoRa_settings_to_byte_array.

```
struct lora_settings lora_settings_from_byte_array(struct byte_array input){
  struct lora_settings from_idiot_name;

  uint16_t freq_f_b = (byte_array_get_element(input, 0)<<8)|(byte_array_get_element(input, 1));
  uint16_t bw_f_b = (byte_array_get_element(input, 3)<<8)|(byte_array_get_element(input, 4));

  from_idiot_name.freq = (float)freq_f_b / (float)10;
  from_idiot_name.power = byte_array_get_element(input, 2);
  from_idiot_name.bw = (float)bw_f_b / (float)10;
  from_idiot_name.cr = byte_array_get_element(input, 5);
  from_idiot_name.preambleLength = (byte_array_get_element(input, 6)<<8)|byte_array_get_element(input, 7);
  from_idiot_name.sf = byte_array_get_element(input, 8);
  from_idiot_name.syncWord = byte_array_get_element(input, 9);

  return from_idiot_name;

}
```

Рисунок 3.22 – Зворотня функція LoRa_settings_to_byte_array.

Структура LoRa_settings містить всі налаштування радіомодему LoRa.

```
struct lora_settings{
      float freq;
      int8_t power;
      float bw;
      uint8_t cr;
      uint16_t preambleLength;
      uint8_t sf;
      uint8_t syncWord;
};
```

Рисунок 3.23 – Структура LoRa_settings.

Опис налаштувань (рис., 3.23):

freq - центральна частота від 410.0 до 493.0 МГц;

power - потужність передачі від -9 до +22 ДБМ (при використанні підсилювача 22 ДБМ підсилюються до 30 ДБМ (1 Вт), що значно підвищує радіус дії);

bw - ширина смуги, LoRa переходить по спектру частот навколо центральної частоти;

cr (coding ratio) - це частка бітів для кодів корекції помилок (число від 5 до 8). Чим більше число - тим більше радіус дії, але менша швидкість передач.

preambleLength - довжина пріамбули. Впливає на розпізнання початку пакету приймачем (від 1 до 65'535). Чим більше значення - тим швидше розпізнає.

sf (spreading factor) - фактор розкиду (значення від 5 до 12). Впливає на радіус дії. Чим більше - тим більше відстань роботи, але й більший час передачі.
syncWord - синхронізаційний байт. Байт, за яким LoRa модеми визначають, що пакет призначений для них.

Функції повідомлень

Радіомесенджер, (73-й стан) є важливим елементом в месенджері й вже згадувався вище. Саме в радіомесенджері реалізована радіокомунікація.

Розберемо основні частини програми зовні циклу.
radiomessenger_active = true; - глобальна зміна активності радіомесенджера (функція radiomessenger_process).
messages_ind_update = true; - оновлення індикатору повідомлень, використовується при оновлені екрану коли з'являються нові повідомлення.
radio.setDio2AsRfSwitch(true); - функція, за командою якої модем LoRa активує підсилювач через пін DIO2.
aes256_radiomessenger.setKey(key, keySize); - структура (об'єкт) типу AES256.

Перевіряємо, якщо не ініціалізовані змінні й повідомлень нуль:
if((messages_count == 0 && message_strings == NULL) && message_types == NULL){
Якщо так, то виділяємо пам'ять:
message_strings = malloc(0);
message_sizes = malloc(0);
message_types = malloc(0);
message_inputs = malloc(0);}

Далі виділяємо пам'ять під буфер (+16 - один підзаголовочний блок) відправки відповідно до введеного з клавіатури тексту:

size_t blocks_count = (kbd.size/16) + 1;
size_t send_buffer_size = (blocks_count + 1)*16;
message_send_buffer = byte_array_create(send_buffer_size);

uint16_t message_send_type = 1; - тип повідомлення, 1 - це тип для короткого тексту (є ще аудіо, у майбутньому будуть додані інші типи).

Запис 16-ти бітного числа, що несе тип повідомлення, до перших двох байтів масиву:
byte_array_replace_element(message_send_buffer,0,message_send_type/256);
byte_array_replace_element(message_send_buffer,1,message_send_type%256);
byte_array_replace_element(message_send_buffer,2,(byte)kbd.size);

Заповнення решти байтів випадковими значеннями:
for (int i = 3; i < 16; i++){
byte_array_replace_element(message_send_buffer,i,radio.randomByte());}

transmitting = true; - індикатор на інтерфейсі.
digitalWrite(LORA_RX_PIN,0); - вимкнути режим прийому.

add_message_from_byte_array(kbd,message_send_type,false); - додаємо повідомлення до масиву байтів. Задаємо зміні:
kbd - масив введеного з клавіатури;
message_send_type - тип повідомлення;
false - задаємо логічне ні, що означає авторство адресанта повідомлення (true - повідомлення адресата).

else{

transmitting = false;
digitalWrite(LORA_RX_PIN,1);
radio.startReceive(RADIOLIB_SX126X_RX_TIMEOUT_INF);}
Якщо не передали й не написали щось на клавіатурі, слухає порт.

transmitting_ind = transmitting; - зміна, за якою горить ліхтарик (світло свідчить про передачу), дорівнюється до змінної передачі.

Перейдемо до коду в циклі.
radiomessenger_process(); - обробляємо прийом пакету по радіомодему, якщо вони прийшли (щоб не загубити під час запису).

Якщо кнопка запису натиснута:
- запускаємо функцію запису audio_record_process();
- audio_path - шлях до файлу куди записуємо;
- додаємо повідомлення типу 2 (аудіо);
- false (наше повідомлення);
- messages_ind_update - виводимо індикатор.
- if(!digitalRead(BUTTON_PIN)){audio_record_process();
- byte_array audio_path = byte_array_create(0);
- add_message_from_byte_array(audio_path,2,false);
- messages_ind_update = true;}

Рисунок 3.24 – індикатор аудіоповідомлень (messages_ind_update).

```
if (messages_ind_update){
druk_messages(texmessages_ind);
texmessages_ind.text = byte_array_create(0);
size_t messages_offset = 0;
for (size_t x = 0; x < messages_count; x++){
```

Якщо індикатор треба оновити, визиваємо функцію druk_messages й друкуємо повідомлення на екран. Як бачимо на рисунку 3.24 - так виглядає вивід текстових й аудіо повідомлень на екран.

Аудіо записується з аналогового мікрофонного підсилювача з 10 бітного АЦП (значення від 0 до 1023). Отримане значення ділиться на 4 до 8 бітів

(значення від 0, до 255). Для економії трафіку частота, в якості звуку в радіокомунікаторі, відіграє значно важливішу роль, ніж розрядність.

Економлячи на розрядності за той самий час радіопередачі відправляється звук більшої частоти.

```
void audio_play_process (){

unsigned long audio_pause = sound_period;
unsigned long audio_timer = micros();
unsigned long audio_counter = 0;

File file;

    TFT_CS_HIGH;
    Sd2Card card;
    card.init(SPI_FULL_SPEED, PIN_SD_CS);
SD.begin(PIN_SD_CS);

MCP4725 MCP(0x60);
  Wire.begin();
  MCP.begin();
  Wire.setClock(800000);
  MCP.setValue(0);
  if (!MCP.isConnected())
  {
    Serial.println("err");

  }

file = SD.open("AUDIO.AUD", FILE_READ);

unsigned long measurement_per_step = 1024;

uint16_t sound;
byte sounds[measurement_per_step];
unsigned long steps = file.size()/measurement_per_step;
byte key[] =
size_t keySize = 32;
byte randbytes[] = {10,11,12,13,14,15,16,17};
byte randbytes2[] = {9,8,7,6,5,4,3,2};
ChaCha chacha;
chacha.setNumRounds (20);
chacha.clear();
chacha.setKey(key,32);
chacha.setIV(randbytes,8);
chacha.setCounter(randbytes2,8);
chacha.hashCore1_0();
chacha.hashCore1_1();
```

Рисунок 3.25 – Функція audio_play_process.

Функція audio_play_process (рис., 3.25) включає в себе такі важливі змінні, як:

audio_pause (sound_period) - період запису;

audio_timer (micros) - час після останнього запису звуку;

audio_counter - лічільник запису звуку до буферу;

file - файл для запису шифрованого звуку;

TFT_CS_HIGH - вимикач SPI шини екрану;

Sd2Card card - об'єкт, що відповідає за microSD картку;

card.init - ініціалізація microSD картки;

SD.begin(PIN_SD_CS) - почати роботу з бібліотекою SD, що відповідає за роботу з файлами;

MCP4725 MCP(0x60) - об'єкт, відповідальний за роботу ЦАПу;

measurement_per_step - розмір буферу;

uint16_t sound - змінна, що відповідає за поточний звук;

byte sounds[measurement_per_step] - буфер;

unsigned long steps = file.size()/measurement_per_step - скільки разів читати й відтворювати звук з файлу;

byte key[] - ключ ChaCha (на скріні не вказаний поточний ключ);

Шифрування

Для захисту комунікацій використовуються алгоритми симетричного шифрування AES (Advanced Encryption Standard) й ChaCha20.

Шифр ChaCha20 змінює елементи один за одним, тому без попереднього елемента або знання кількості загублених елементів розшифрувати шифр, на даний момент, неможливо. ChaCha20 шифрує звук, у той час як AES шифрує текст. ChaCha20 на практиці проводить шифрування в 8 разів швидше за AES, тому саме він вибраний для шифрування звуку. Шифр AES на платі Arduino не встигає проводити всі необхідні шифрування.

Код відкритої (Open-source software) бібліотеки шифру ChaCha20 був модифікований й вказаний в ДОДАТКУ. В код були додані паралельні функції void hashCore для розбиття процесу хешування на більш короткі по часу

підпроцеси. Це необхідно для швидкого кодування звуку, бо Arduino не встигає в один момент обробляти всю функцію хешування (64 виміри Arduino не встигає зробити при визові однієї функції).

AES - це симетричний шифр, що шифрує блоками по 16 байт. В даному випадку використовує ключ на 256 біт (32 байти). Через фіксовані за розміром блоки AES оперує масивами 4 на 4 байт (так званими станами, чи англійською state). В даному шифрі кожен байт замінюється за ключем по 8-ми бітній таблиці - цей метод також відомий як SubBytes.

Окрім стандартного шифрування AES, до нього доданий CBC (Cipher Block Chaining, рис., 3.26) - режим зчеплення блоків шифротексту. Проблема шифра AES без модифікацій в тому, що ті самі частини шифрованого тексту відповідають відповідним розшифрованим частинам якщо ключ той самий - такої криптографічної вразливості можна позбутися при використанні CBC.

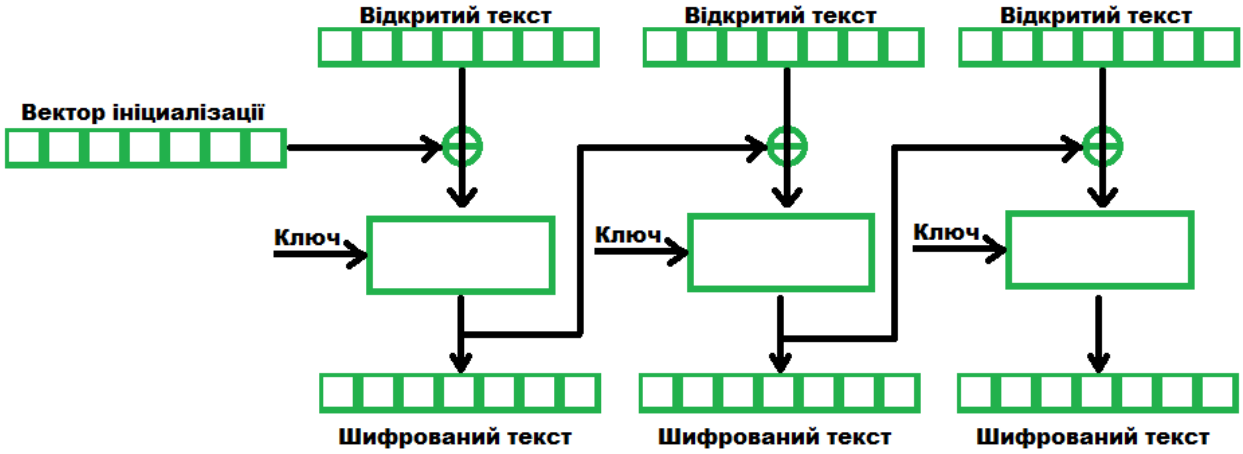


Рисунок 3.26 – Шифрування з ланцюгом блоків (CBC).

3.3 Тестування та аналіз результатів роботи

Тестування компонентів пристрою проводилось в спеціалізованих підпрограмах, які мали наступний вигляд (рис., 3.27, 3.28).

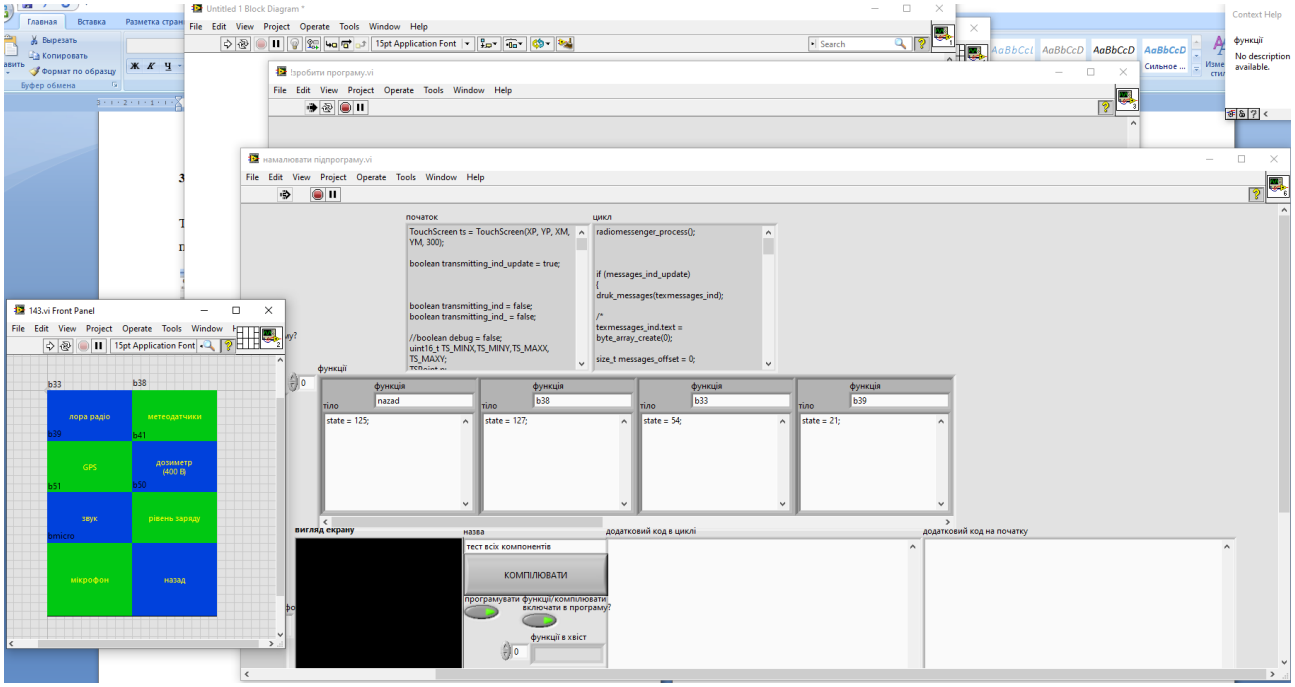


Рисунок 3.27 – Вигляд меню підпрограм тестування в АрдПрогу.

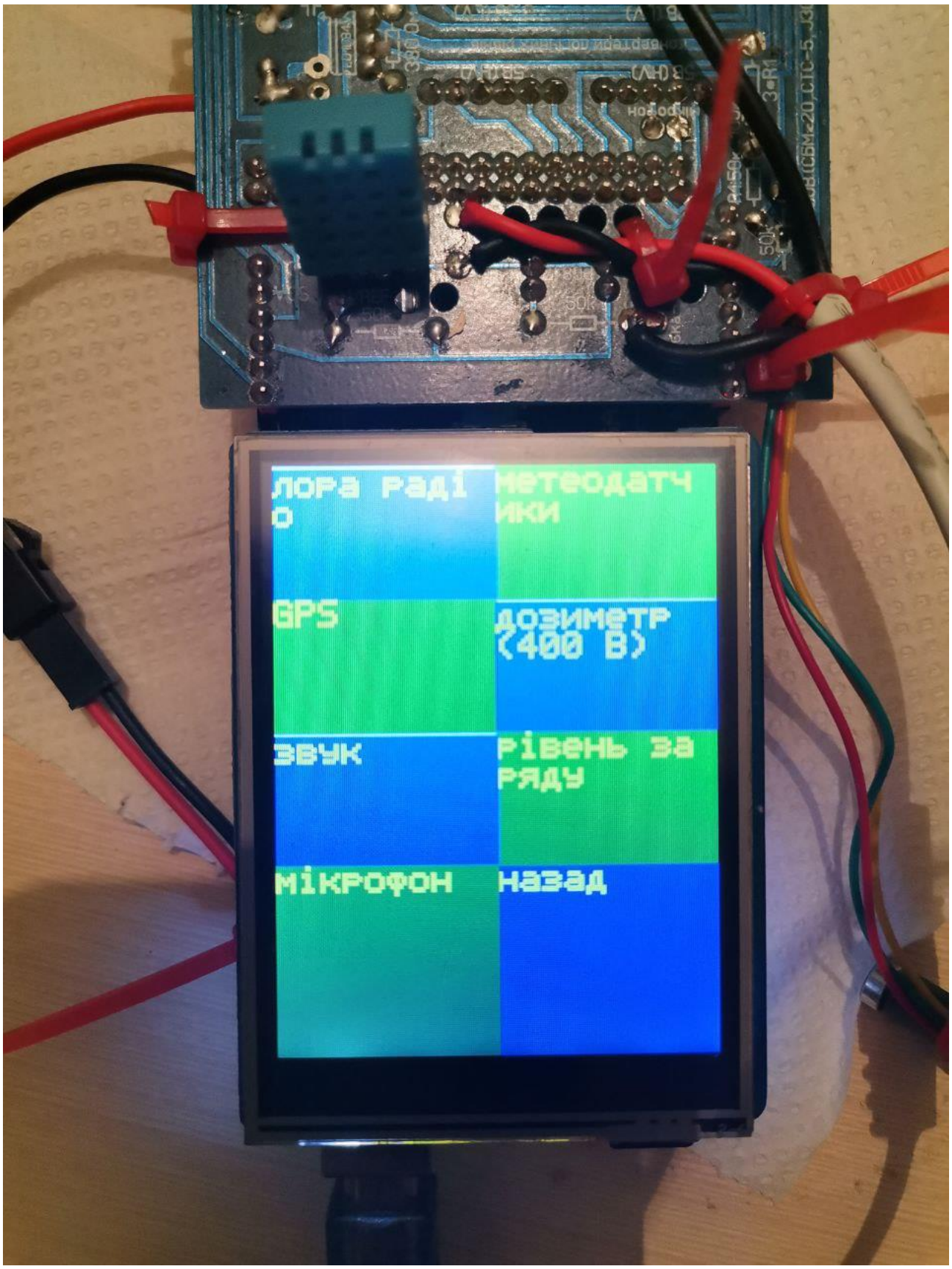


Рисунок 3.28 – Вигляд меню підпрограм тестування в комунікаторі.

Натискання кожної кнопки призводить до зміни змінної "state", що відповідає за вибір підпрограми. До кожного стану (state) є свій унікальний код.

Почнемо з тесту працездатності модема LoRa - 54-й стан.

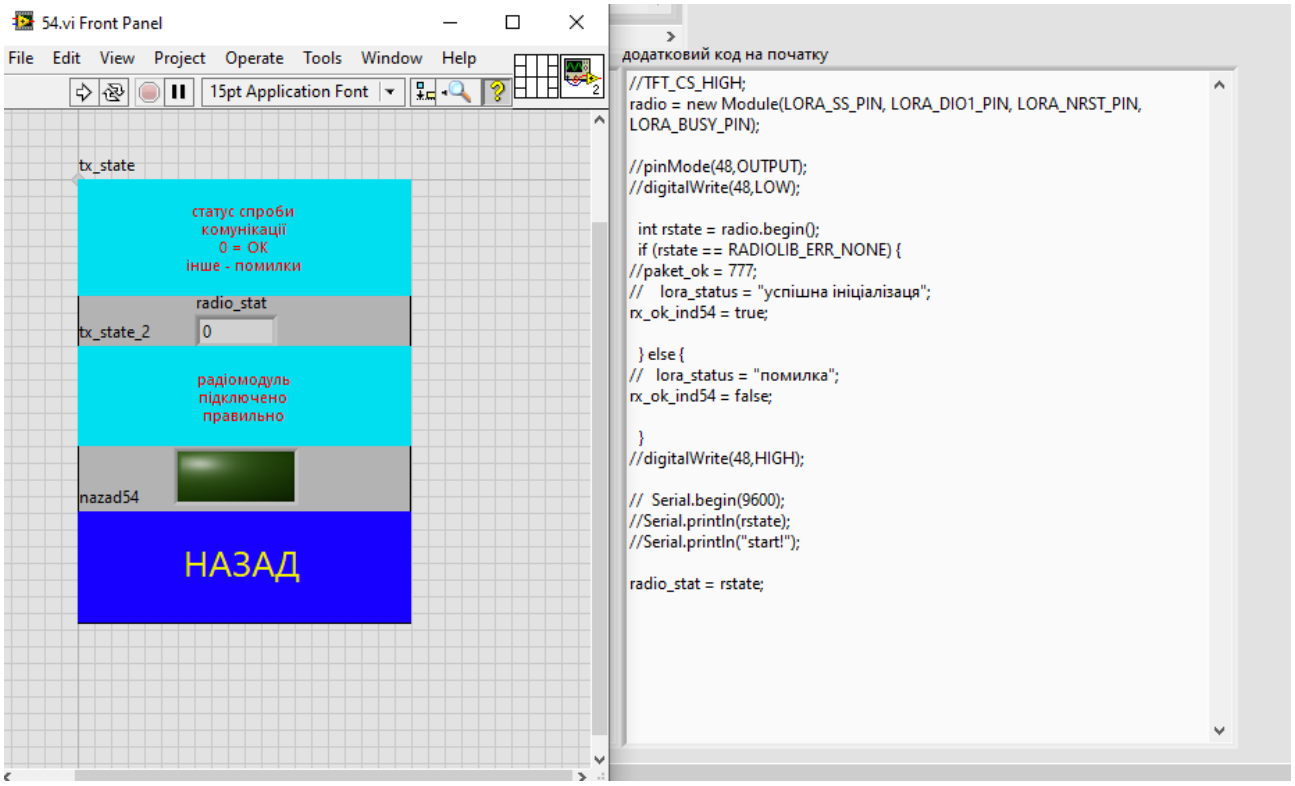


Рисунок 3.29 – Вигляд меню підпрограм тестування в комунікаторі.

Даний стан викликає функцію radio.begin(); (бібліотека RADIOLIB), за результатом виклику якої отримуємо код помилки або її відсутності (0 при відсутності помилок, булевий індикатор видає зелене світло). Код помилки виводиться на стан radio_stat.

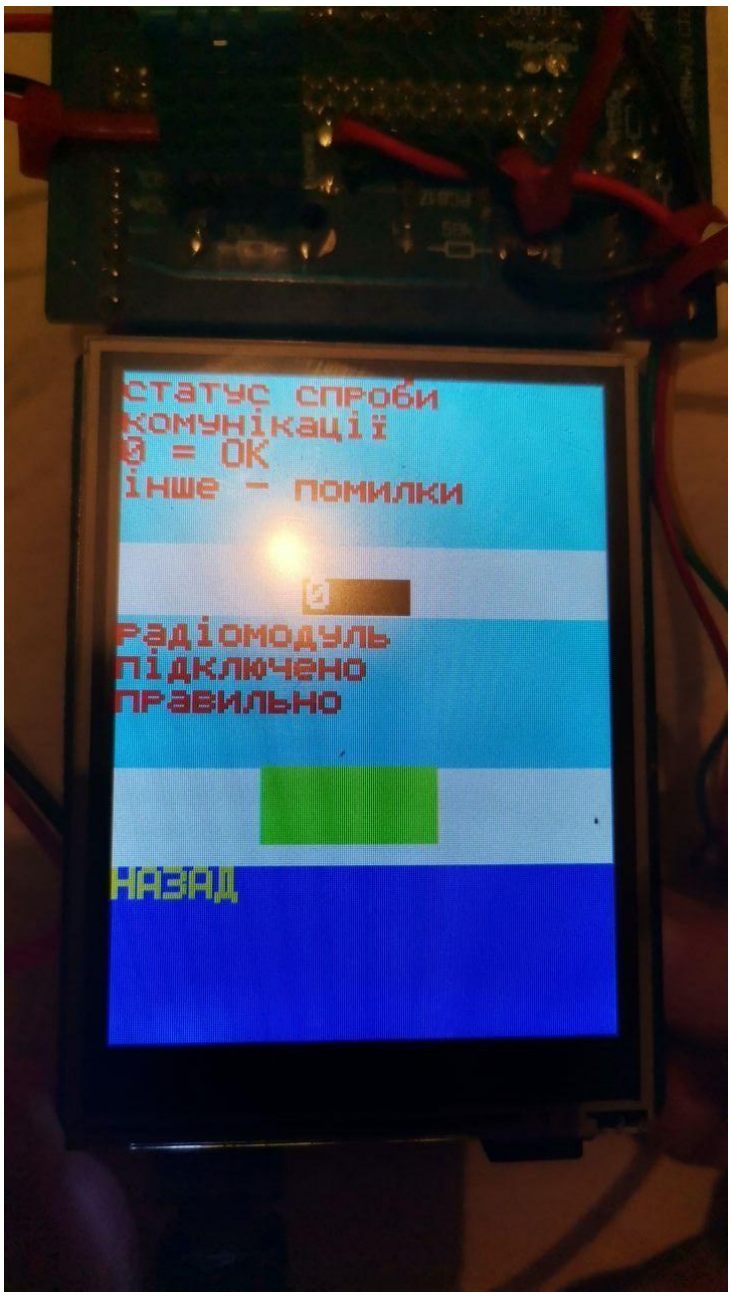


Рисунок 3.30 – Підпрограма тестування радіомодему LoRa.

Як бачимо на рисунку 3.30, підпрограма тестування видає результат про успішну роботу LoRa радіомодему. Це свідчить про те, що живлення радіомодему й підключення до плати Arduino працюють правильно.

ВИСНОВКИ

У кваліфікаційній магістерській роботі, першочергово, був проведений інформаційний огляд. Інформаційних огляд можна поділяється на огляд предметної області, аналіз аналогів й постановку задачі.

Наступним етапом був вибір методів рішення. Зупинившись на кращому варіанті реалізації, вибравши плату Arduino й радіомодем LoRa, як основу пристрою, почалась розробка прототипу. Закінчивши етап розробки прототипу, було написано відповідне програмне забезпечення.

Після створення ПЗ й реалізації функціоналу в комунікаторі, були проведені тестування задля реалізації функціоналу. Завершуючи розробку комунікатору, зазначу наступне:

- робочий прототип комунікатору зібраний;
- комунікатор запрограмований й успішно працює;
- комунікатор протестований і всі знайдені баги виправлено.

Прототипи комунікатору працюють, передають зашифровану інформацію й використовують різні модулі для її збору. Попит на розробку вже є у військових і деякі цивільні також зацікавились комунікаторами. Тому можна вважати, що інформаційно-комунікаційна технологія забезпечення захищеного зв'язку успішно розроблена і має світле майбутнє. У найближчий час технологія додатково буде модифікована, задля отримання нового функціоналу і підвищення якості.

Завершуючи кваліфікаційну роботу, хочу висловити слова подяки своїм друзям: фізику Глібу Положію й радіомонтажнику Владу Родіну, що приймали участь в створенні і тестуванні комунікаторів.

СПИСОК ЛІТЕРАТУРИ

1. "E22-400M30S User Manual" – 2018. – Режим доступу до ресурсу: https://www.ebyte.com/
2. “SX1268 Long Range, Low Power, sub-GHz RF Transceiver“ – 2019. – Режим доступу до ресурсу: https://www.semtech.com/
3. “Arduino.cc - official arduino site. Documentation“. Режим доступу до ресурсу 5 вересня 2022: https://docs.arduino.cc.
4. “Manual Adafruit io HTTP API Adafruit IO API Reference“. Режим доступу до ресурсу: https://io.adafruit.com/api/docs/ – 2022.
5. “Eli Crow, Designer & Developer“.Режим доступу до ресурсу 18 листопада 2022: https://elicrow.com/ .
6. “QMesh: A Synchronized, Flooded Mesh Network Protocol for Voice“ – Dan Fay, KG5VBY TAPR DCC Режим доступу до ресурсу: 2020 – https://www.particle.io/blog/how-to-build-a-wireless-mesh-network/
7. “Understanding LoRa Adaptive Data Rate“ Режим доступу до ресурсу: December 2019 – www.semtech.com
8. “Multimediavia LoRa“ Режим доступу до ресурсу: 23 жовтня 2022 https://www.researchgate.net/publication/319647571_Transfer_of_Multimedia_Data_via_LoRa
9. “M17 Project”, https://m17project.org/, Режим доступу до ресурсу: квітня 29, 2022
10. "FREEDV: OPEN SOURCE AMATEUR DIGITAL VOICE”, Режим доступу до ресурсу: https://freedv.org/, accessed 15 серпня, 2022
11. “GoTenna Mesh”, https://gotennamesh.com/products/mesh, Режим доступу до ресурсу: 30 квітеня, 2022
12. “What is Meshtastic?”, https://www.meshtastic.org/, Режим доступу до ресурсу: 30 квітеня, 2022.
13. “Disaster.radio: “, https://disaster.radio/, Режим доступу до ресурсу: 30 квітеня, 2022.

14. “New Packet Radio”, https://hackaday.io/project/164092-npr-new-packet-radio, Режим доступу до ресурсу: 30 квітеня, 2022.
15. “VEMesh IoT Products”, https://virtual-extension.com/products/iot-products/, Режим доступу до ресурсу: 4 листопада, 2022.

ДОДАТОК

Бібліотека ChaCha.h, модифікована.

```
/*
 * Copyright (C) 2015 Southern Storm Software, Pty Ltd.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 */
```

```
#ifndef CRYPTO_CHACHA_h
#define CRYPTO_CHACHA_h

#include "Cipher.h"

class ChaChaPoly;

class ChaCha : public Cipher
{
public:
    explicit ChaCha(uint8_t numRounds = 20);
    virtual ~ChaCha();

    size_t keySize() const;
    size_t ivSize() const;

    uint8_t numRounds() const { return rounds; }
    void setNumRounds(uint8_t numRounds) { rounds = numRounds; }

    bool setKey(const uint8_t *key, size_t len);
    bool setIV(const uint8_t *iv, size_t len);
    bool setCounter(const uint8_t *counter, size_t len);

    void encrypt(uint8_t *output, const uint8_t *input, size_t len);
    void decrypt(uint8_t *output, const uint8_t *input, size_t len);

    void clear();
```

```
    static void hashCore(uint32_t *output, const uint32_t *input, uint8_t rounds);

    void hashCore1();
void hashCore1_0();
void hashCore1_1();
void hashCore1_2();
void hashCore1_3();
void hashCore1_4();
void hashCore1_5();
void hashCore1_6();
void hashCore1_7();
void hashCore1_8();
void hashCore1_9();
void hashCore1_10();
void hashCore1_11();
void hashCore1_12();
void hashCore1_13();
void hashCore1_14();
void hashCore1_15();
void hashCore1_16();
void hashCore1_17();
void hashCore1_18();
void hashCore1_19();
void hashCore1_20();
void hashCore1_21();
void hashCore1_22();
void hashCore1_23();
void hashCore1_24();
void hashCore1_25();
```

```
void hashCore1_26();
void hashCore1_27();
void hashCore1_28();
void hashCore1_29();
void hashCore1_30();
void hashCore1_31();
void hashCore1_32();
void hashCore1_33();
void hashCore1_34();
void hashCore1_35();
void hashCore1_36();
void hashCore1_37();
void hashCore1_38();
void hashCore1_39();
void hashCore1_40();
void hashCore1_41();


private:
    uint8_t block[64];
    uint8_t stream[64];
    uint8_t rounds;
    uint8_t posn;

    void keystreamBlock(uint32_t *output);

    friend class ChaChaPoly;
```

```
    //additional stream for next block hashing in parallel
    uint8_t stream1[64];

};

#endif
```