Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

«Допущено до захисту»
В.о. зав.кафедрою БІСТ
Ольга МЕЛКОЗЬОРОВА

_____

« » 2024 р.

Пояснювальна записка
до кваліфікаційної роботи бакалавра
на тему: «Безпека вбудованих систем»

оцінка « »
Голова ЕК
Олександр ЛЕМЕШКО

_____

Керівник: к.т.н. Ольга МЕЛКОЗЬОРОВА
Рецензент: д.т.н. Віктор КРАСНОБАЄВ
Виконавець: студент групи КБ-41

Богдан ВАЛЮХ

Харків – 2024

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра містить 61 сторінку, 55 рисунків, один додаток та перелік з 17 використаних джерел.

Метою дипломної роботи є аналіз безпекової ситуації на ринку вбудованих систем доступу, розробка власної системи доступу на основі сучасних embedded рішень, демонстрація потенційних вразливостей вбудованих систем та надання рекомендацій про організацію структури типової вбудованої охоронної системи доступу.

Предмет включає в себе вивчення різних підходів щодо отримання несанкціонованого доступу до об’єкту, що охороняється розробленою охоронною системою доступу, а також способів усунення вразливостей, що можуть призвести до успішної реалізації описаного вище.

Об’єктом роботи є проект вбудованої охоронної системи доступу, що піддається тестуванню кількома методами вторгнення.

В результаті роботи було виявлено та проаналізовано критичні вразливості у вбудованій системі доступу, що можуть бути використані зловмисниками для отримання доступу до об’єкту, що охороняється.

Результати розробки та аналізу можуть бути використані для кращого розуміння можливих недоліків та вразливостей охоронної системи доступу. при прийнятті рішення про придбання чи самостійну розробку такої.

Ключові слова: КІБЕРБЕЗПЕКА, ВБУДОВАНІ СИСТЕМИ, ОХОРОННІ СИСТЕМИ, КОНТРОЛЬ ДОСТУПУ, STM32.

ABSTRACT

The explanatory note to the bachelor's thesis contains 61 pages, 55 figures, one appendix and a list of 17 references.

The purpose of the thesis is to analyze the security situation in the market of embedded access systems, develop an own access system based on modern embedded solutions, demonstrate potential vulnerabilities of embedded systems and provide recommendations on the organization of the structure of a typical embedded security access system.

The subject includes the study of various approaches to obtaining unauthorized access to the object protected by the developed security access system, as well as ways to eliminate vulnerabilities that can lead to the successful implementation of the above.

The object of the work is a project of an embedded security access system that is subjected to testing by several intrusion methods.

As a result of the work, critical vulnerabilities in the embedded access system were identified and analyzed that can be used by attackers to gain access to the protected object.

The results of the development and analysis can be used to better understand the possible shortcomings and vulnerabilities of the security access system when deciding whether to purchase or independently develop one.

Keywords: CYBERSECURITY, EMBEDDED SYSTEMS, SECURITY SYSTEMS, ACCESS CONTROL, STM32.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

МК	–	Мікроконтролер
ШІМ	–	Широтно-Імпульсна Модуляція
ПЗ	–	Програмне Забезпечення
AES	–	Advanced Encryption Standard
IoT	–	Internet Of Things
I2C	–	Inter-Integrated Circuit
SPI	–	Serial Peripheral Interface
USB	–	Universal Serial Bus
UART	–	Universal Asynchronous Receiver-Transmitter
SDA	–	Serial Data
SCL	–	Serial Clock

ВСТУП

Вбудовані системи стали необхідною складовою нашого повсякденного життя, впливаючи на наші звички, зручність та безпеку. Сьогодні вони використовуються в безлічі різних сфер – від систем охорони та автоматизації розумного будинку до медичних пристроїв підтримання життєдіяльності та розгалужених комплексів керування транспортом. З ростом кількості пристроїв та галузей їх застосувань зростає і важливість забезпечення їх безпеки – кібербезпеки вбудованих систем.

Вбудовані системи, які зазвичай мають обмежені ресурси, такі як обсяг пам'яті, обчислювальну потужність та енергоспоживання, викликають особливі виклики у забезпеченні їх кібербезпеки. Кібербезпека вбудованих систем є одним з найголовніших, часто критичним аспектом, адже вони нерідко стають бажаною ціллю для атак з боку зловмисників, що може бути виражено як в крадіжці персональних даних, так і знищенню критично важливої інфраструктури. В гонитві за «оптимізацією» (як часу, так і коштів) нерідко допускаються фатальні безпекові помилки. Часто можна почути заголовки «3 мільйони зубних щіток були використані при DDoS атаці» чи «В мережу витекли записи з домашніх камер тисяч громадян *вставити країну*», і це лише вершина айсбергу. Якщо зламом «домашніх» вбудованих систем настільки успішно займаються семі-професіонали лише для незначної вигоди, то можна лише уявити на що готові піти зацікавлені в промисловому шантажі чи підриві національної безпеки групи осіб.

У цій роботі будуть розглянуті найпоширеніші, на погляд автора, вразливості, що притаманні широкому колу повсякденних пристроїв «інтернету речей», а також можливі стратегії, рішення та підходи, що можуть забезпечити безпеку та надійність вбудованих систем.

Мета цієї роботи – дослідити сучасний стан кібербезпеки вбудованих систем, перевірити безпекові рішення в популярній IoT галузі та надати рекомендації щодо запобігання потенційним загрозам в майбутньому.

1 ЦИФРОВІ СИГНАЛИ У ВБУДОВАНИХ СИСТЕМАХ

1.1 Загальні поняття

Сигнал в загальному випадку являє собою певну змінну фізичну величину, що використовується для передачі даних - будь-який потік інформації можна охарактеризувати як коливання цієї фізичної величини. В контексті вбудованих пристроїв поняття сигналу найчастіше використовується по відношенню до електричних (найчастіше, коливання, зміна з часом, електричного струму) та радіосигналів (процес розповсюдження інформації за допомогою радіохвиль). Також, іноді можна зустрітись з використанням оптичних сигналів у вбудованих системах, але сфера використання таких сигналів незначна в порівнянні з описаними вище.

В залежності від функції, що описує параметри сигналу, можна виділяють [1]:

- Аналогові (безперервні) сигнали, у яких кожен з параметрів описується функцією часу і безперервною множиною можливих значень;
- Дискретні сигнали, що описуються функцією відліків, взятих в конкретний проміжок часу, і які можуть приймати лише скінченну кількість значень;
- Квантовані аналогові сигнали (розбивка всієї області значень сигналу на рівні, кількість яких має бути представлена в числах заданої розрядності від 0 до N-1, відстань між ними називається кроком квантування);
- Цифрові (квантовані дискретні) сигнали, що описуються функцією дискретного часу та скінечною кількістю значень. Цифрові сигнали представляють собою послідовність нулів та одиниць (двійкова система числення), що репрезентують квантове значення сигналу на кожному проміжку часу, представлене у вигляді цілого числа.

Сьогодні, в якості «корисних» сигналів в сучасних ком’ютерних системах, найчастіше виступають саме цифрові сигнали. Передача в зрозумілому, як для відправника, так і для отримувача, бінарному форматі є основною складовою швидкої та (майже завжди) більш «loseless» передачі інформації.

1.2 Протоколи передачі даних

Передача інформації, безумовно, є найважливішою складовою вбудованих систем. Простий мікроконтролер наодинці може виступати перемикачем у електричному колі або ШІМ-регулятором яскравості підсвітки в домі, але чи це все, на що здатні сучасні МК?

Візьмемо за приклад задачу – потрібно створити систему парктроніків для нової моделі авто. З вимог – максимально швидке та точне звукове відображення поточної дистанції від датчику відстані в салоні автомобіля. Для пробного старту є все потрібне – сучасний мікроконтролер, точний датчик та кілька метрів кабелю. Постає проста, на перший погляд, задача – передавати інформацію про відстань від сенсора (Slave'а) до головного керуючого пристрою – мікроконтроллера (Master'а). Але як? Тут і приходять на допомогу протоколи комунікації, або, по іншому, протоколи передачі даних.

Протокол передачі даних – це деяка система правил, якої повинні дотримуватись усі учасники діалогу для успішної та «lossless» передачі інформації – такими правилами можуть бути структури даних чи способи і методи синхронізації.

На сьогодні існує досить велика кількість протоколів, усі зі своїми плюсами та недоліками. Розглянемо декілька найпопулярніших з них:

1.2.1 USB

Широко використовуваний протокол USB (Universal Serial Bus) в розрізі являє собою двопровідний послідовний інтерфейс, що дозволяє підключати до 127 (2^7 – 1) пристроїв одночасно, причому підключення може відбуватись в будь-який відрізок часу (підтримка Plug'n'Play). USB протокол надсилає та отримує дані послідовно між головним хостом та зовнішньо підключеними пристроями за допомогою сигнальних ліній D+ (Data+) та D- (Data-). Крім цього, кожен Slave девайс має на борту VCC та GND лінії для, власне, живлення його роботи. Схема виводів ліній/пінів USB зображена на рис. 1.1.

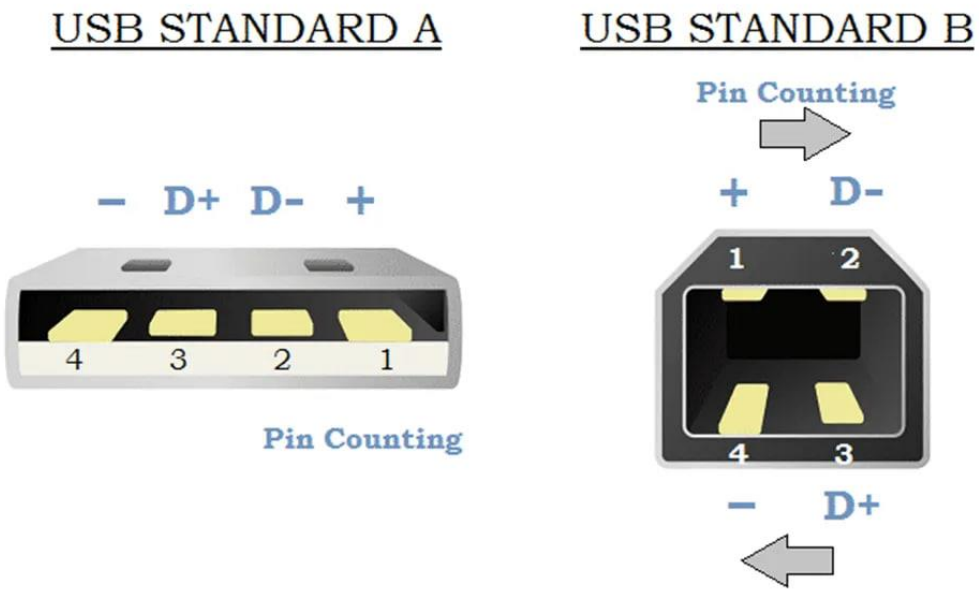

Рисунок 1.1 – Виводи пінів USB

1.2.1.1 USB комунікація

USB — це «опитувана» шина, де USB-хост ініціює всі обміни даними. Клієнт на USB-хості зьерігає дані в буферах, але не має кінцевих точок. USB-хост і USB-девайс мають окремі рівні. З’єднання між рівнями є нічим іншим, як логічними хост-девайс інтерфейсами між кожним горизонтальним шаром. Між логічними з’єднаннями дані передаються за допомогою каналів (Ріре’ів) [2]. Схема логічних з’єднань між хостом та девайсом показана на рис. 1.2.

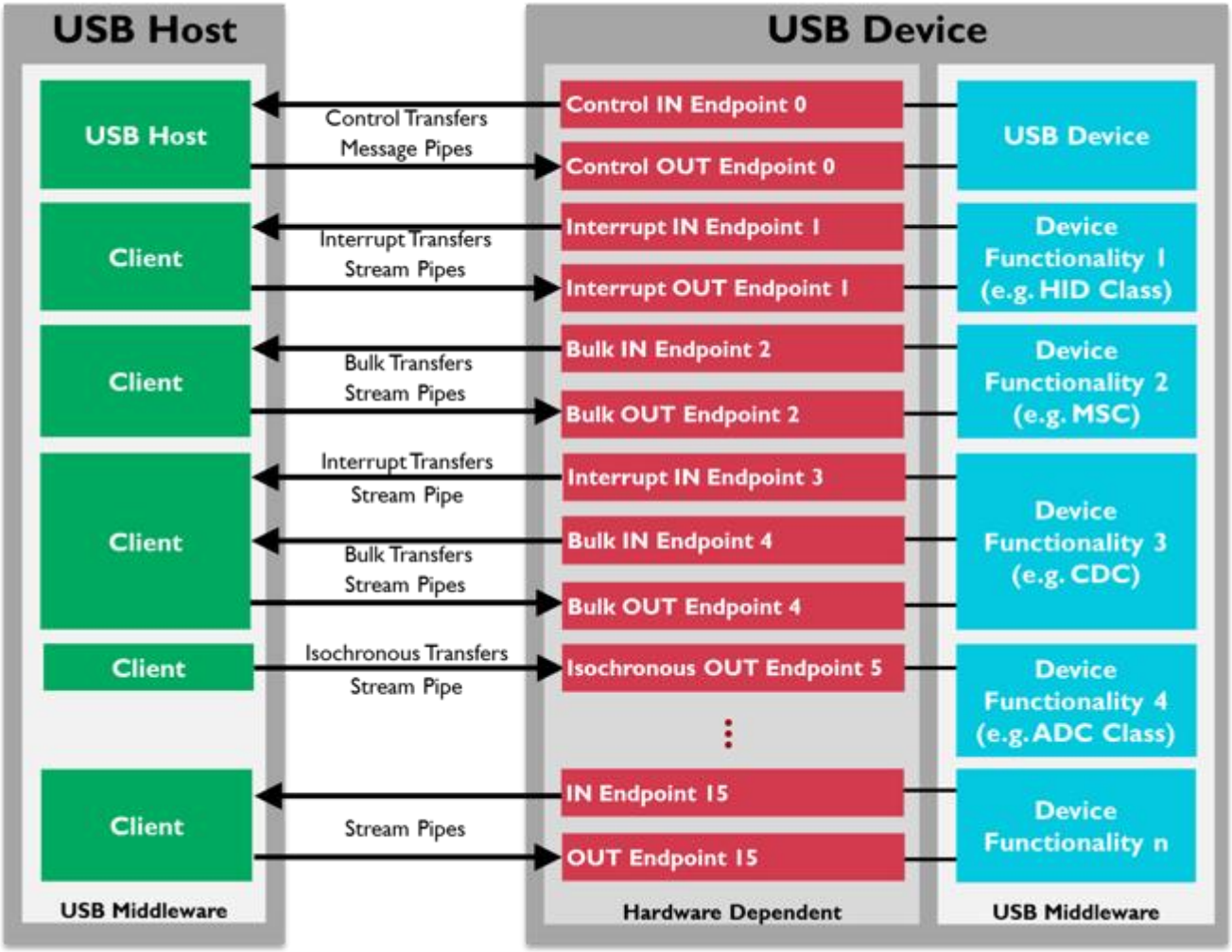

Рисунок 1.2 – Логічні з'єднання між USB-хостом та USB-девайсом

1.2.1.2 Транзакції

Дані передаються в так званих транзакціях. Зазвичай, вони (транзакції) складаються з трьох пакетів [3]:

- Токен, пакет «маркерів» - це заголовок, що визначає тип і напрямок транзакції, адресу пристрою та кінцеву точку;
- Пакет даних – частинка даних що, власне, передаються. Пакети даних складається з одного байту (8 біт), причому спочатку передається найменш значущий біт (Least Significant Bit, LSB);
- Пакет «рукостискання» – остаточний статус транзакції, її завершення.

Структура транзакцій зображена на рис. 1.3.

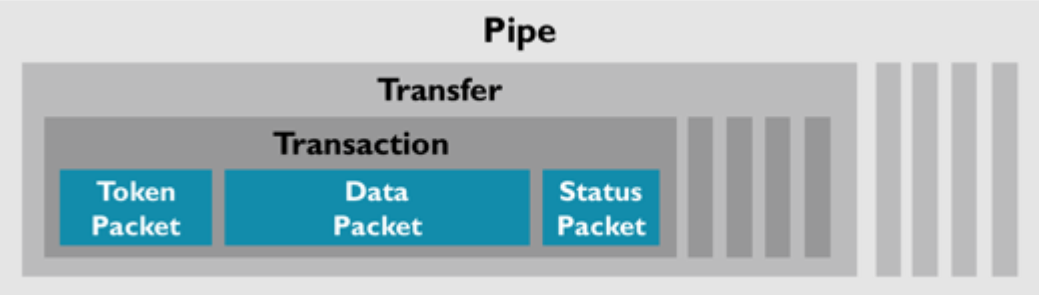

Рисунок 1.3 – Структура транзакції в USB

Під час транзакції дані передаються або з USB-хосту на USB-девайс, або з USB-девайсу на ISB-хост. Напрямок передачі вказується в першому пакеті (пакеті маркерів), який надсилається з USB-хосту. Потім, пристрій надсилає пакет даних або вказує, що не має даних для передачі. Наприкінці адресат відповідає пакетом рукостискання, вказуючи, чи була остання передача успішною [3]. Модель пакету в USB показана на рис. 1.4.

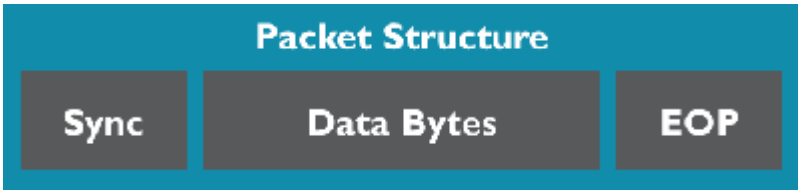

Рисунок 1.4 – Структура/модель пакету в USB

1.2.1.3 Пакети

Пакети можна розглядати як найменший елемент передачі даних. Кожен пакет передає цілу кількість байтів з поточною швидкістю передачі (рис 1.5):

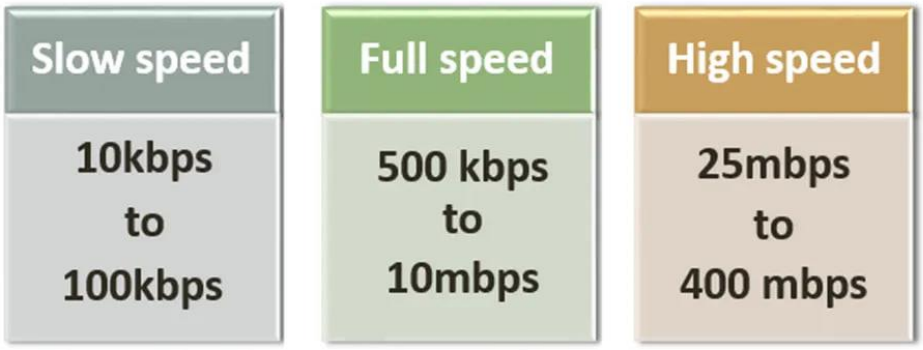

Рисунок 1.5 – Можливі варіанти швидкостей USB

Пакети починаються з паттерну синхронізації, за яким ідуть байти даних пакета , за яким йде End-Of-Packet (EOP) сигнал кінця пакету. Усі паттерни USB-пакетів передаються найменш значущими бітами попереду. До і після пакету шина знаходиться у стані очікування (рис. 1.6).

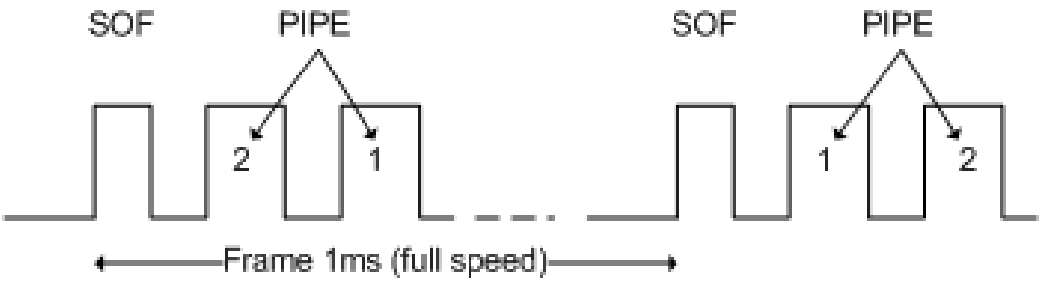

Рисунок 1.6 – Start-of-Frame пакет

Спеціальним пакетом є Start-of-Frame (SOF) пакет, який розбиває шину USB на часові сегменти. Кожному каналу виділяється слот у кожному фреймі. SOF пакет надсилається кожну 1мс на «Full speed» з'єднаннях. На швидкості «High speed» фрейм тривалістю 1мс ділиться на 8 мікрофреймів, по 125мкс кожен. Start-of-Frame пакет надсилається на початку кожного мікрофрейму з тим самим номером фрейму, який в свою чергу інкрементується кожні 1мс [3].

1.2.1.4 Переваги та недоліки USB-інтерфейсу

Доволі детально розглянувши будову USB протоколу потрібно підсумувати його «за» та «проти» в контексті використання у вбудованих системах:

+ Швидкодія та ємність потенційної передачі даних (особливо в нових стандартах);
+ Невелика загальна собівартість – в мережі сотні, якщо не тисячі варіацій на тему впровадження USB протоколу в ембеддед проект разом з низькою ціною на необхідні компоненти;
+ Plug'n'Play – суттєвий плюс, якщо потрібна підтримка «гарячої» заміни Slave девайсів.

- Для роботи з декількома девайсами потрібен потужний хост – як в плані швидкодії, так і в плані можливостей вихідного живлення, адже усі девайси будуть живитись саме з хоста;
- Неможливість спілкування девайсів між собою – передача даних можлива лише між хостом та девайсом, варіант коли один з девайсів стає хостом в USB протоколі не є можливим.

1.2.2 SPI

SPI (Serial Peripheral Interface), як слідує з назви, це послідовний чотирьохпровідний синхронний інтерфейс, що призначений для повнодуплексного обміну даними між хостом та Slave девайсами. В основному SPI використовують для передачі даних між мікроконтроллером та зовнішньою пам’яттю (QuadSPI), дисплеями, Ethernet або CAN контролерами, АЦ/ЦАП та іншими високошвидкісними датчиками. Для синхронізації та для передачі даних на шині SPI виведено дві окремі лінії[4]. Схема з’єднання пристроїв на шині показана на рис. 1.7:

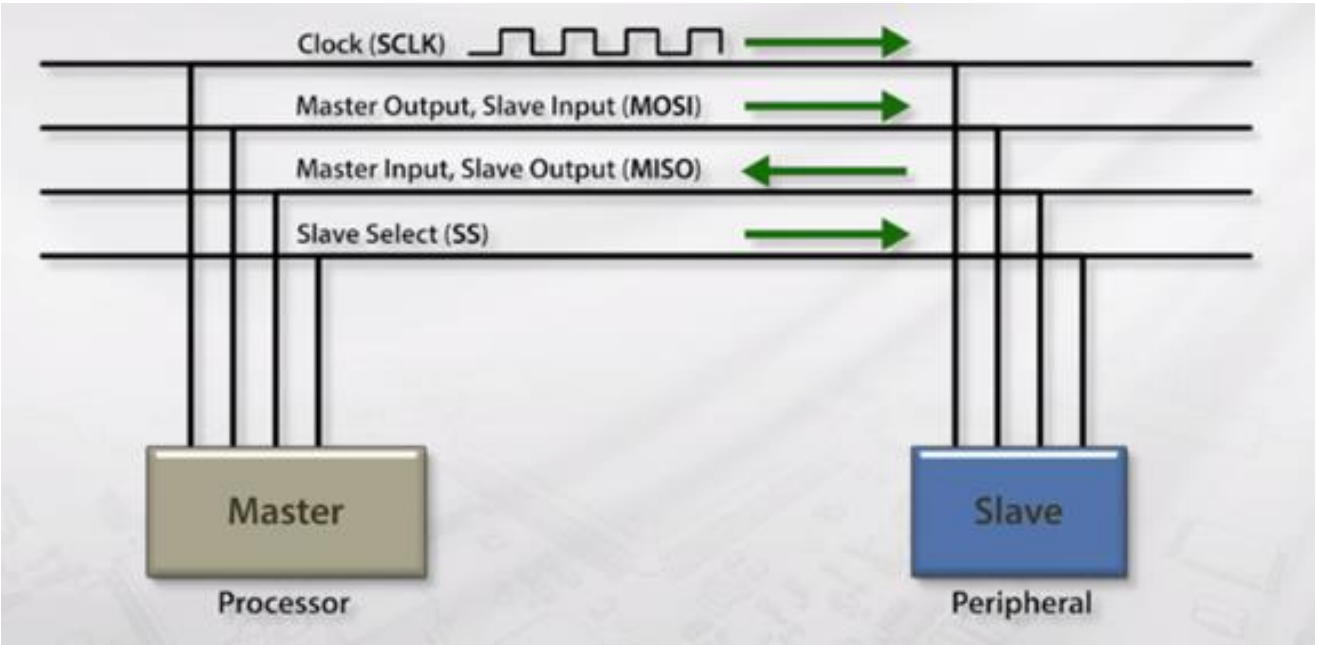

Рисунок 1.7 – Схема з'єднання головного та підлеглого пристроїв

1.2.2.1 SPI комунікація

По аналогії з USB (і не тільки, як ми дізнаємось далі) – основним (головним) на шині є ведучий пристрій/мастер-девайс/хост (найчастіше – мікроконтролер). Генерація ним тактового сигналу (SCLK) дозволяє синхронізувати передачу даних в обох напрямках – як вхідних, так і вихідних (виходи MISO – Master In Slave Out та MISO – Master Out Slave In відповідно).

Також на SPI шині присутній сигнал CS (Chip Select, на рис.1.7 позначений як SS – Slave Select), що призначений для інформування пристрою про необхідність встановлення зв’язку. Зазвичай це активний низький (логічний нуль) сигнал, який стає високим (логічна одиниця) при завершенні сеансу зв'язку. За наявності кількох підлеглих пристроїв керуючому необхідно мати окремий вихід для кожного з підлеглих[5].

Варто зазначити, що на одній шині можуть бути декілька потенційних ведучих пристрої (два мікроконтролера, наприклад) і вони можуть по черзі ставати то ведучим, то підлеглим. Найголовніше в подібній реалізації – відсутність колізії між ними на проміжку часу – в конкретний момент часу лише один пристрій на щині може бути головним. Схема підключення кількох підлеглих пристроїв зображена на рис. 1.8.

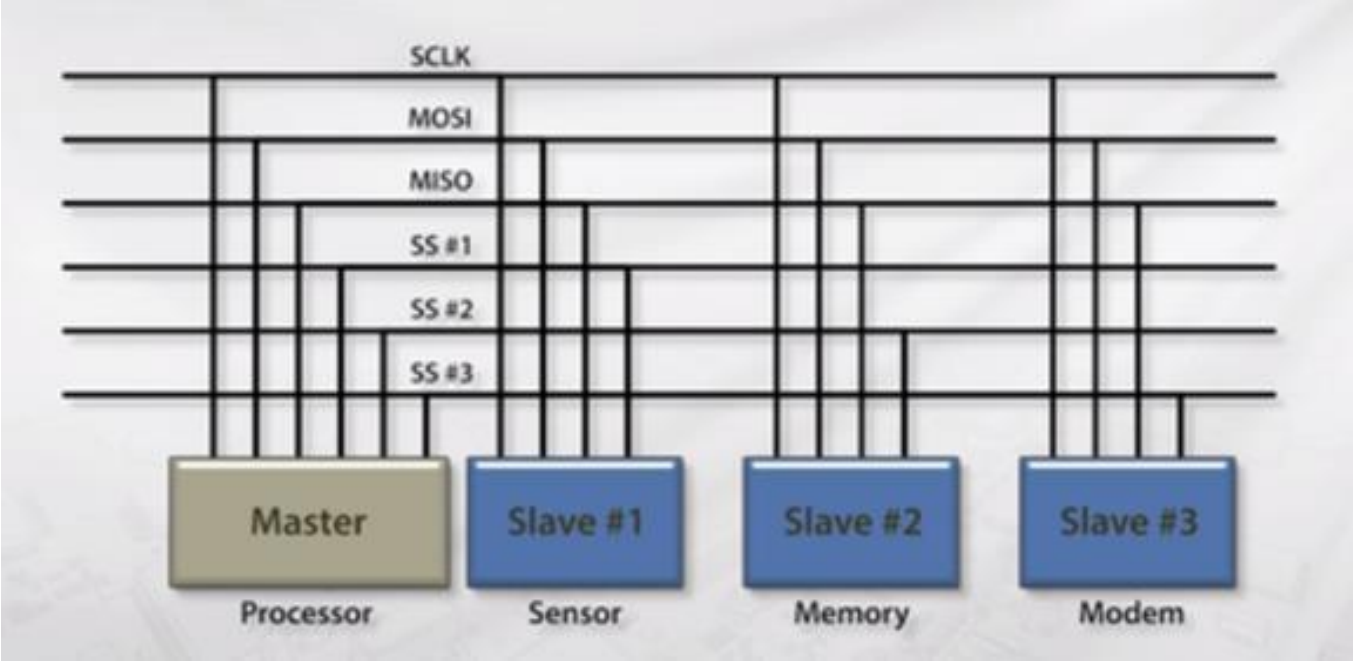

Рисунок 1.8 – Кілька підлеглих пристроїв в управлінні керуючого пристрою

Передача потоку даних контролюється ведучим пристроєм, оскільки він є джерелом тактового сигналу. Дані передаються побітово напротязі одного сигналу синхронізації – одночасно від ведучого до підлеглого та від підлеглого до ведучого. Таким чином в обох напрямках передається один байт даних за один сеанс передачі. Тобто, можна зробити висновок, що SPI являє собою повнодуплексний зв'язок [5].

При наявності на шині SPI кількох підлеглих пристроїв – активним може бути лише той, і тільки той, пін CS (SS на схемах) якого було підтягнуто до високого рівня. При створенні колізії (кілька пристроїв на шині мають встановлений пін CS) – поведінка може бути невизначеною [5].

Що стосується апаратного рівня – передача являє собою два зсувних регістра по одному байту для ведучого та підлеглого пристроїв, що утворюють цикл передачі. Дані, як правило, зсуваються старшим бітом наперед (MSB, Most Significant Bit) (рис. 1.9)[5] .

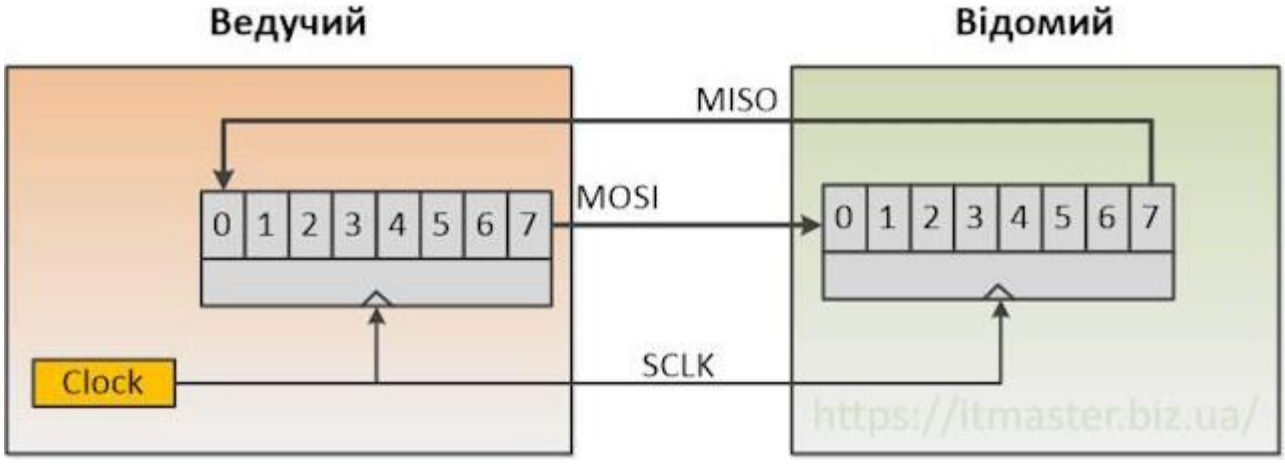

Рисунок 1.9 – Регістри зсуву на головному та ведучому пристроях

Передача даних здійснюється пакетами довжиною вісім біт (один байт). Перед стартом дані для передачі поміщаються у регістри зсуву. Одразу після завершення попереднього пункту ведучий пристрій починає генерувати імпульси синхронізації для взаємної передачі та обміну даними. Побітова передача даних триває до тих пір, допоки не буде передано весь байт даних. За потреби передачі додаткових даних зсувні регістри перезавантажуються і процес починається спочатку [5]. Візуальна репрезентація процесу передачі зображена на рисунку 1.10:

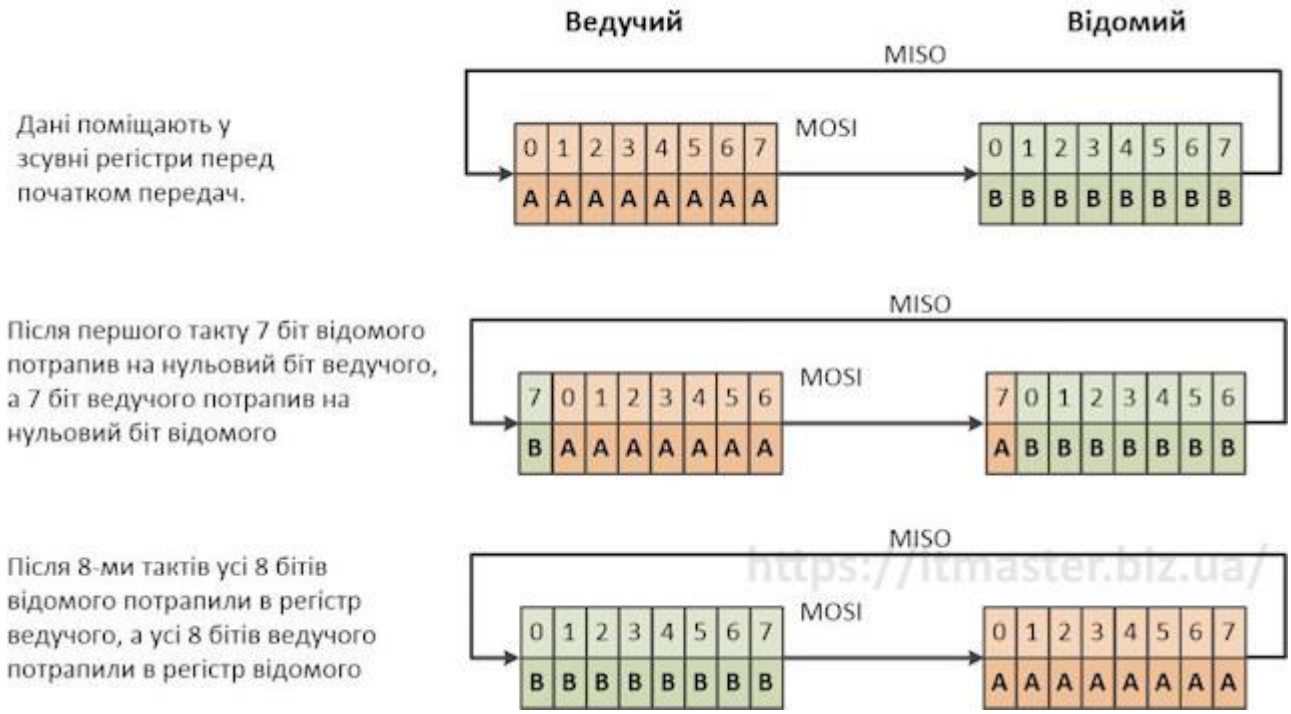

Рисунок 1.10 – Процес передачі у SPI

Все ж, зазвичай, SPI працює в режимі передачі головним пристроєм команд з прийомом від периферійного відповідей на них. Нижче показана діаграма сигналів, де ведучий пристрій висилає команду 80d, а підлеглий пристрій відсилає у відповідь число 70d (рис 1.11) [5].

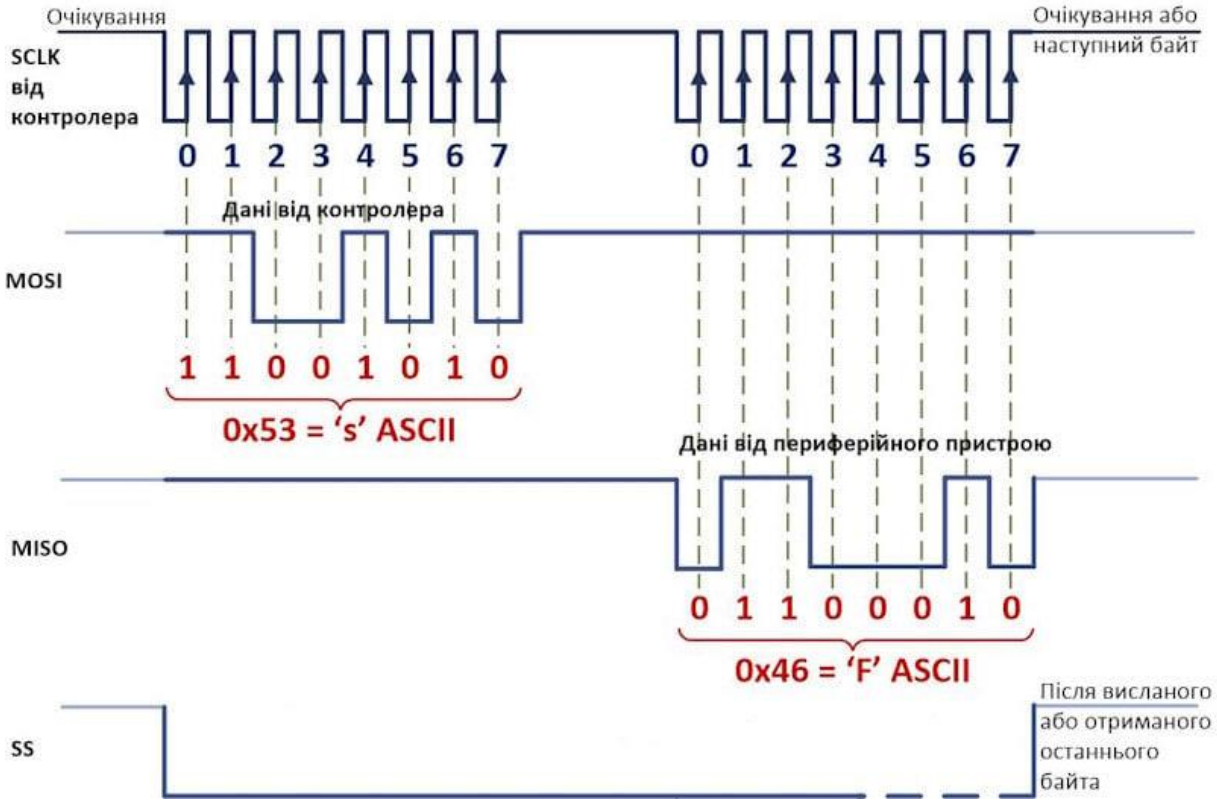

Рисунок 1.11 – Діаграма спілкування МК та девайсу

1.2.2.2 Переваги та недоліки SPI

+ Повнодуплексний зв’язок;
+ Висока швидкість передачі при загальній простоті інтерфейсу;
+ Можливість підключення нескінчено великої кількості пристроїв (головне щоб вистачило виходів на керуючому пристрої);
+ Широкі можливості для спілкування двох потенційно ведучих пристроїв;
+ Низьке електроспоживання.
- Відсутнє підтвердження отримання даних;
- Необхідність великої кількості виходів для підключення великої кількості пристроїв.

1.2.3 I2C

Інтерфейс I2C являє собою двопровідний синхронний послідовний інтерфейс, розроблений компанією Philips. Його призначення – обмін інформацією між мікроконтролером та девайсами на відносно невеликих відстанях.

I2C є доволі повільним інтерфейсом зі швидкістю передавання інформацією близько 400 Кб/сек. Найширше його використовують для зв’язку з датчиками, що не потребують великої швидкості передачі та знаходяться близько до Master девайсу.

1.2.3.1 I2C комунікація

Інформація в I2C передається по шині, що, як було сказано вище, складається з двох провідників – лінії даних SDA (Serial Data Line, послідовна лінія даних) та лінії тактування SCL (Serial Clock Line, послідовна лінія тактів) [6].

На шині I2C повинні бути присутні ведучий (або ведучі – I2C дозволяє наявність декількох Master пристроїв одночасно) пристрій та від одного до 127 (2^7 - 1) периферійних пристроїв.

Варто зазначити, що стандарт I2C передбачає можливість використання 10-бітної адреси, що потенційно збільшує кількість одночасно підключених пристроїв до 1023 (2^10 – 1), але на практиці переважна більшість периферійних пристроїв підтримують лише 1 – 4 унікальних 7-бітні адреси.

Приклад схеми підключення пристроїв показаний на рис. 1.12:

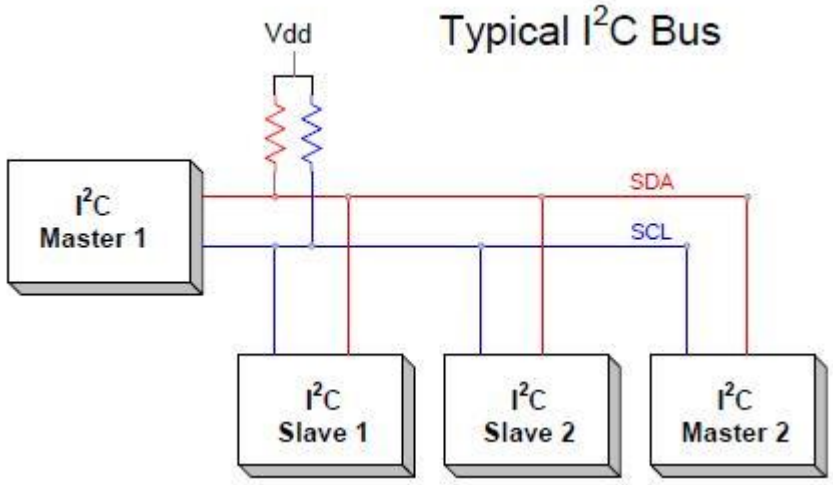

Рисунок 1.12 – Приклад схеми підключення пристроїв на шині I2C

Для спілкування ведучого пристрою з периферійними пристроями перший повинен згенерувати тактовий сигнал на шині, звернутись до потрібного Slave пристрою за його унікальною адресою (обов'язкова унікальність адрес власне і обумовлює обмеження в 127 девайсів на одній шині) і, в залежності від встановленого біта R/W записати/зчитати дані. Периферійні пристрої можуть відповідати лише на запит Master пристрою зі своєю унікальною адресою – самостійно ініціалізувати передачу пристрій не можуть [7].

Візуальна репрезентація передачі показана на рисунку 1.13:

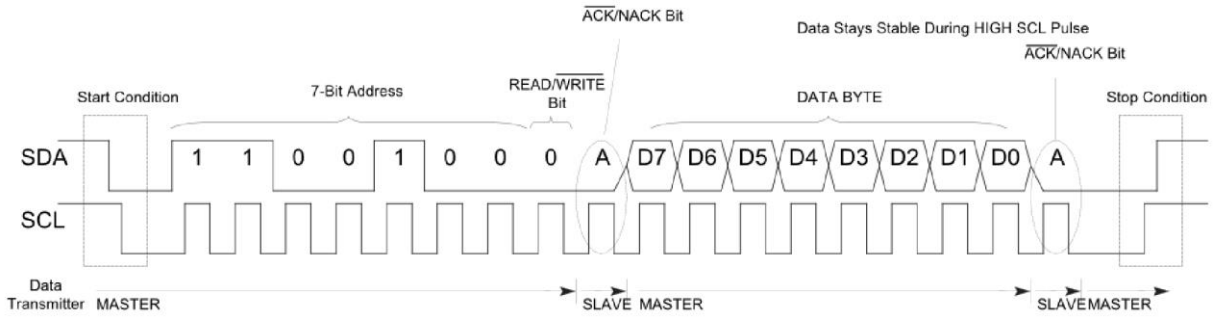

Рисунок 1.13 – Діаграма передачі I2C протоколу

В I2C протоколі повідомлення діляться на два типи кадрів – кадр унікальної адреси веденого пристрою (за яким даний пристрій переходить в режим очікування

даних, якщо передано 0 в біті R/W, чи в режим відправки даних, якщо передано 1 в біті R/W) та один або безліч кадрів даних, в якому, в залежності від встановленого біта R/W, ведучий чи периферійний пристрій передають дані отримувачу (рис. 1.14)[6].

Рисунок 1.14 – Приклад повідомлення в протоколі I2C

Інформація по I2C шині передається у вигляді пакетів, що містять адресу периферійного девайсу, номер (адресу) регістру, до якого звертаються та дані для передачі від ведучого до підлеглого (або навпаки) [6].

Може виникнути ситуація, коли швидкість надсилання даних від головного контролера буде перевищувати максимальну можливу швидкість обробки цих даних – момент, коли дані ще не готові (наприклад, периферійний пристрій ще не завершив цикл аналого-цифровеого перетворення) або часовий проміжок, коли виконання попередньої операції ще не було завершено [6].

У цьому випадку більшість підлеглих пристроїв будуть виконувати так зване «розтягування тактування» (Clock Stretching) – пристрій утримує низький рівень лінії тактування вже після того, як ведучий звільнив її. Охарактеризувати це можна одним реченням – девайс виграє час перед наступною передачою для завершення виконання попередньої (рис. 1.15) [6].

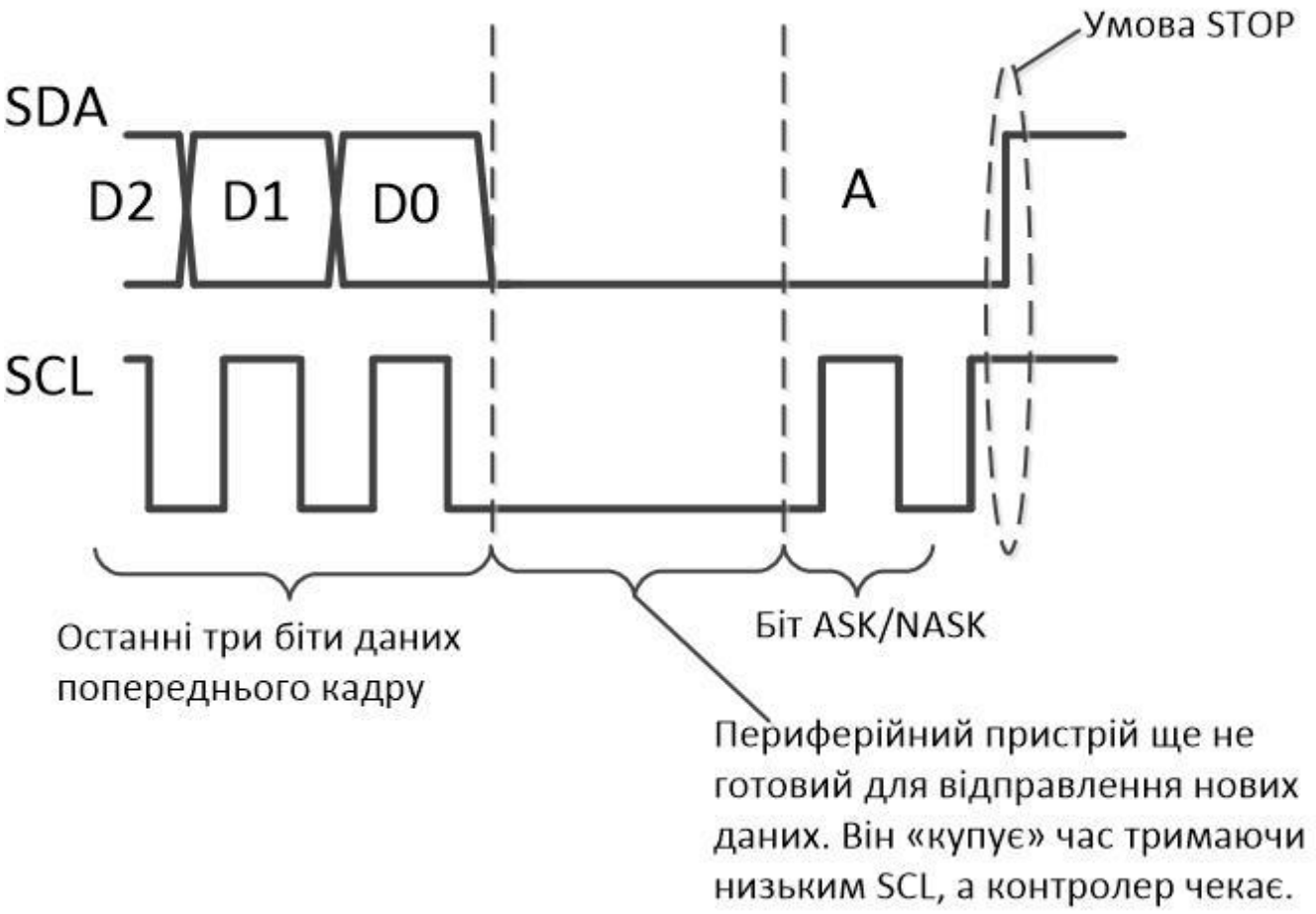

Рисунок 1.15 – Приклад "розтягування тактування" в I2C

1.2.3.2 Переваги та недоліки I2C інтерфейсу

+ Потрібно всього два провідника для забезпечення зв’язку між пристроями;
+ Можливості роботи декількох Master пристроїв на одній шині;
+ Можливість підключення нескінчено великої кількості пристроїв (головне щоб вистачило виходів на керуючому пристрої);
+ Підтримка «Hot Plug» для периферійних пристроїв.
- Обмежена швидкість та дальність передачі даних;
- Обмежена кількість пристрої (зазвичай не більше 127) на одній шині;
- Необхідність чіткого та постійного моніторингу помилок – один «зависший» пристрій здатний «покласти» усю шину без відповідних заходів у відповідь.

1.2.4 UART/USART

UART – Universal Asynchronous Receiver-Transmitter, що перекладається як Універсальний Асинхронний Приймач-Передавач. На відмінну від попередньо розглянутих інтерфейсів UART є, власне, асинхронним інтерфейсом передачі даних. Це означає, що прийом та передача окремих бітів не «вирівнюються» синхроімпульсами, що одночасно є як перевагою, так і недоліком. Одиниця від нуля відрізняється виключно по часу між перепадами рівнів сигналу, який задається швидкістю передачі даних (Baud Rate, кількість бітів за секунду) [10].

З плюсів асинхронної передачі даних можна виділити загальну простоту протоколу, мінімальні апаратні затрати на обмін даними та можливість роботи в (окрім також симплексного та напівдуплексного) повнодуплексному режимі [10].

Найважливішим недоліком асинхронного методу передачі інформації є жорстка залежність вірності отриманих даних від правильності встановленого Baud Rate'у на обох пристроях. Сьогодні, найпоширенішими є швидкості 4800, 9600, 19200, 57600 і, найпоширеніший, 115200 бітів на секунду. Похибка швидкостей передбачена стандартом UART і може складати близько 5% [9].

З боку апаратної частини контролер UART являє собою регістр зсуву, що є основним методом перетворень між паралельною та послідовною формами передачі. [9].

Схема роботи UART контролера показана нижче, на рисунку 1.16:

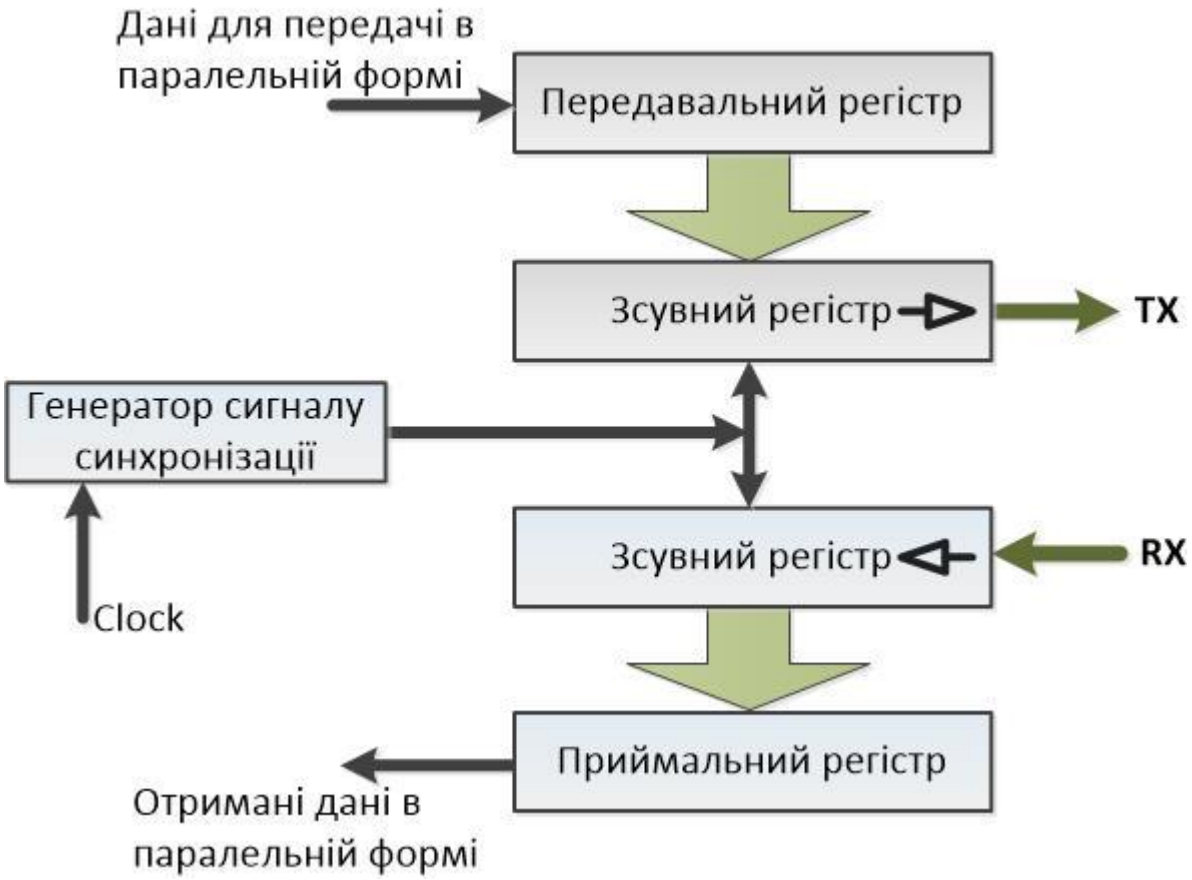

Рисунок 1.16 – Схема роботи UART контролера

1.2.4.1 Комунікація UART

Інтерфейс UART складається з приймача та передавача. Стандарт не задає ведучий та ведений пристрій, оскільки обидва девайси мають однакові права на рівні інтерфейсу – розробник повинен сам, на рівні софту, вирішити питання розподілу обов’язків між пристроями.

Уся шина UART складається з двох провідників – від піна Tx (Transmit) першого пристрою до піна Rx (Receive) другого пристрою і від піна Tx другого пристрою до піна Rx першого пристрою, тобто під’єднання відбувається навхрест (рис. 1.17).

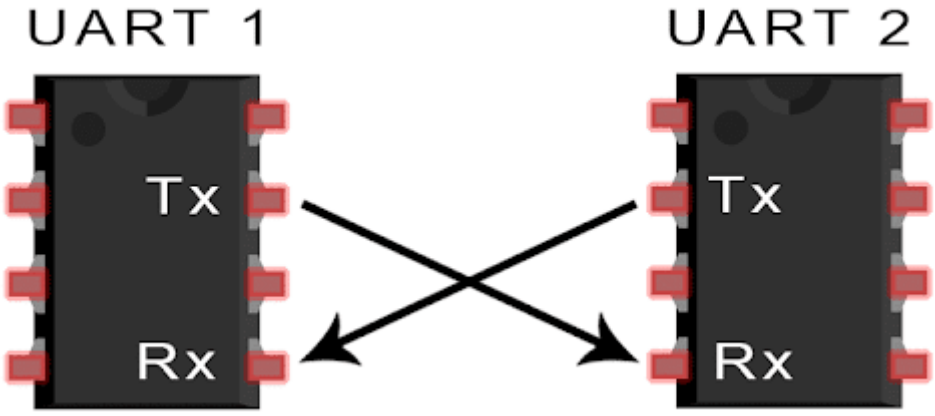

Рисунок 1.17 – Схема підключення UART

З точки зору зв'язку в UART – передача відбувається в, так званих, фреймах. Кожен фрейм складається з бітів – фрагментів інформації, яка послідовно відправляється по лінії. Ці біти включають початковий біт, біти даних, один біт парності (може бути відсутнім, його наявність оговорюється між девайсами завчасно) та один-два стоп-біти [10].

Необов'язковий біт парності дозволяє приймачу примітивно перевірити, чи правильно отримані дані. Тип паритету, як і наявність цього біта, оговорюється завчасно та може бути як парним, так і непарним.

Кожен біт, як нам відомо, може нести в собі одиницю (висока напруга) або нуль (низька напруга). Біт, як одиниця даних, повинен передаватись у суворо відвенений для нього проміжок часу (що, фактично, являється тайм-слотом) (рис. 1.18).

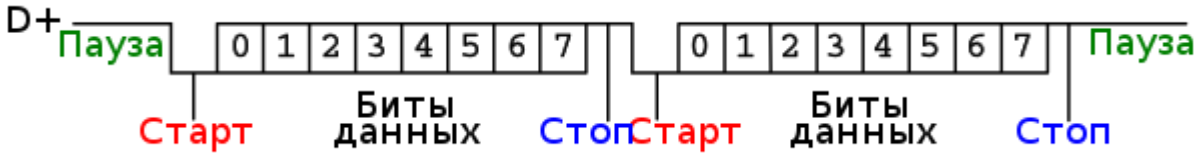

Рисунок 1.18 – Передача байтів в UART

В момент простою шини – на лінії встановлюється та зберігається логічна одиниця. Встановлення низького рівню (логічного нуля) – це сигнал приймачу про початок передачі даних (байту), обов'язково з дотриманням завчасно заданих часових відрізків. Після передачі усього байту (після сьомого (8) біта) відправник

надсилає стоп-сигнал у вигляді логічної одиниці, як сигнал приймачу про завершення передачі даних.[10].

Рядок з повідомленням, що передається по UART можна побачити нижче, на скріншоті з осцилографа (рис. 1.19):

Рисунок 1.19 – Фрагмент повідомлення на шині UART на осцилографі

1.2.4.2 Переваги та недоліки UART

+ Потрібно всього два провідника для забезпечення зв’язку між пристроями;
+ Простота роботи з інтерфейсом;
+ Можливість роботи в повнодуплексному режимі.
- Можливість обміну інформацією лише між двома пристроями (в теорії можливе створення мережі з одним ведучим та багатьма веденими пристроями, але через архітектурні особливості UART – це можливо лише у вигляді «надсилати однакові дані на декілька UART-USB перетворювачів одночасно з одного девайсу»);
- Жорсткі часові обмеження при прийомі даних;
- Все ж, нижча загальна надійність протоколу в порівнянні з синхронними протоколами (є синхронний USART).

2 РОЗРОБКА ОХОРОННОЇ СИСТЕМИ ДОСТУПУ

2.1 Постановка задачі

Трохи дізнавшись про наявні сигнали та існуючі протоколи їх передачі – можемо перейти до розробки власної системи контролю доступу.

Стоїть задача створення невеликого пристрою з клавіатурою, що буде виступати в якості цифрового замку – відкривати двері при введенні правильного коду доступу та закривати їх при постановці на охорону. Також, в систему потрібно інтегрувати сповіщаючий пристрій, що буде відтворювати звукову індикацію при постановці/знятті з охорони, а також сигналізувати про деяку кількість невдалих спроб введення паролю.

При розробці також варто врахувати побажання мати постійну індикацію статусу системи на основному пристрої і можливість підключення декількох дверей до однієї системи.

Якнайшвидша передача системи замовнику є пріоритетом.

2.2 Вибір апаратної платформи та розробка схеми підключення

2.2.1 Апаратна платформа

Так як швидкість розробки стоїть першим пунктом у вимогах до розробника - в якості основи апаратної платформи було вибрано ARM сімейство мікроконтролерів компанії ST STM32, через свою легкість в початкових налаштуваннях та досвід роботи з ним в команді розробників.

Для прототипування головного пристрою було використано наявні на складі плати STM32F407G-DISC1. В якості основи для сигналізуючого пристрою (надалі – сирени) було вибрано плату розробника STM32VLDISCOVERY на мікроконтролері STM32F100RB (рис. 2.1).

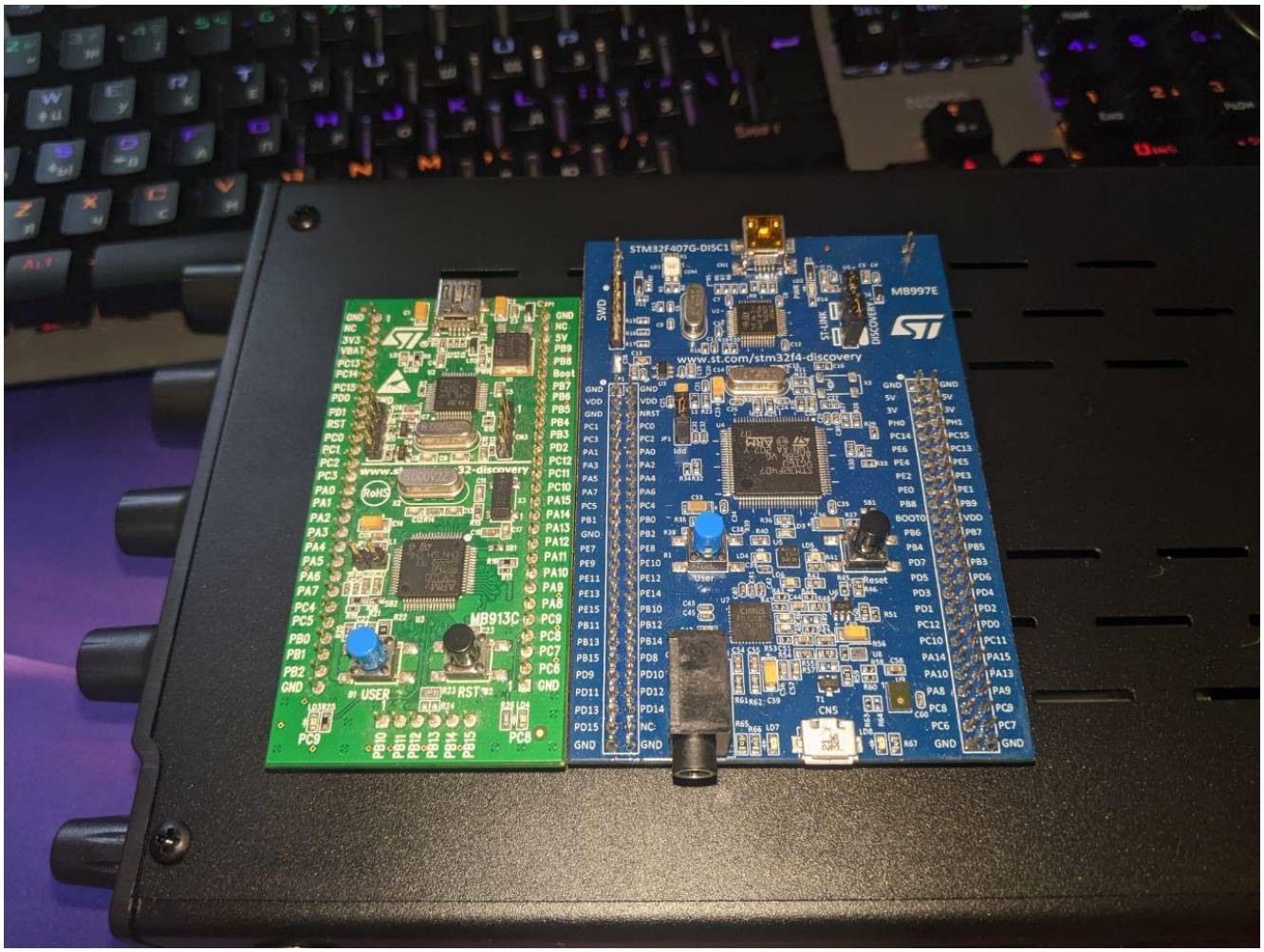

Рисунок 2.1 – STM32VLDISCOVERY (зліва) і STMF407G-DISC1 (справа)

В якості перемикача «відкрито/закрито» було вибрано посилений модуль реле, що керується одним IO виходом (без урахування виводів живлення). Такий спосіб реалізації дозволить підключати до вихідної лінії декілька механічних пристроїв відмикання замків, що цілком відповідає наданим вимогам.

В якості засобу оптоіндикації було вибрано вбудовані в плату світлодіоди (з перспективою заміни на адресний світлодіод на WS2812) і недорогий та надійний РК дисплей LCD16x2. Керування дисплеєм, задля простоти реалізації та швидшого завершення розробки, буде здійснюватись через I2C модуль розширення виводів PCF8574. Підключивши усі 16 виводів дисплею до модуля можна досягти повної доступної функціональності LCD16x2, використовуючи при цьому в 7(!) разів менше необхідних керуючих пінів (без врахування виводів живлення).

В якості клавіатури було вирішено використати просту та надійну 12-ти кнопкову мембранну клавіатуру, що працює за принципом замикання контактів колонок та рядків (рис. 2.2):

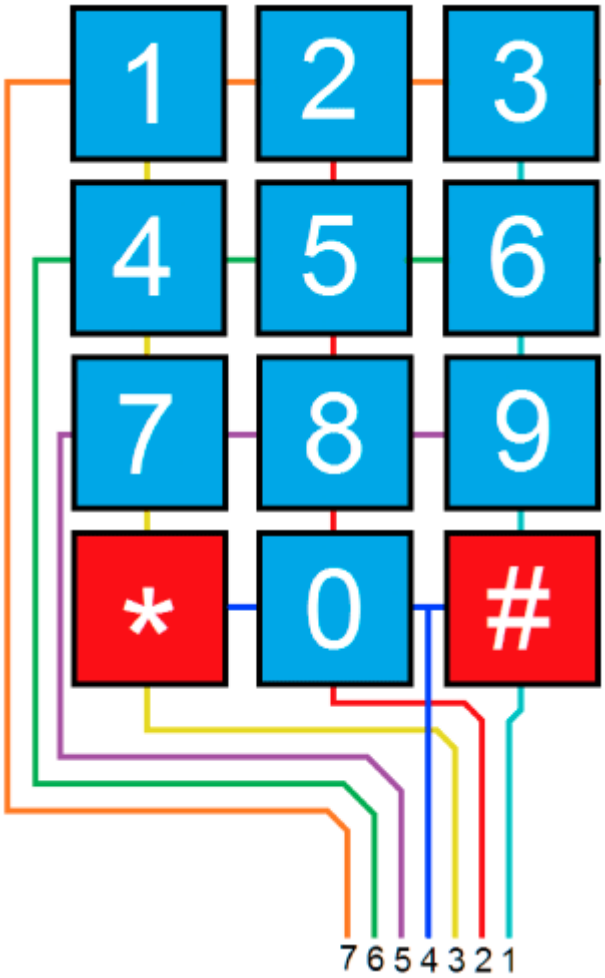

Рисунок 2.2 – Розводка клавіатури [11]

На зображенні вище видно – при подачі живлення на лінії колонок 1-3 і натисканні якої-небудь кнопки – на лінії її стовпця з’явиться логічна одиниця. Перемикаючи живлення на лініях колонок і слідкуючи за логічним рівнем на лінії стовпця під питанням можна точно визначити яка саме з клавіш була натиснута. Був вибраний саме такий варіант реалізації через наявність 4 окремих ліній EXTI0-EXTI3 в основному керуючому пристрої – при наявності меншої кількості незалежних (лінія виключно для одного номера піна) EXTI ліній можна скористатись або зворотнім способом моніторингу (подавати живлення на 4 лінії рядків, і слідкувати за лініями стовпців), але це вносить хоч і невеликі, та все ж затрати по часу на перевірку (в найгіршому випадку – 4 цикли IO замість 3).

Рисунок 2.3 є демонстрацією електронних модулів, що будуть використані при розробці системи:

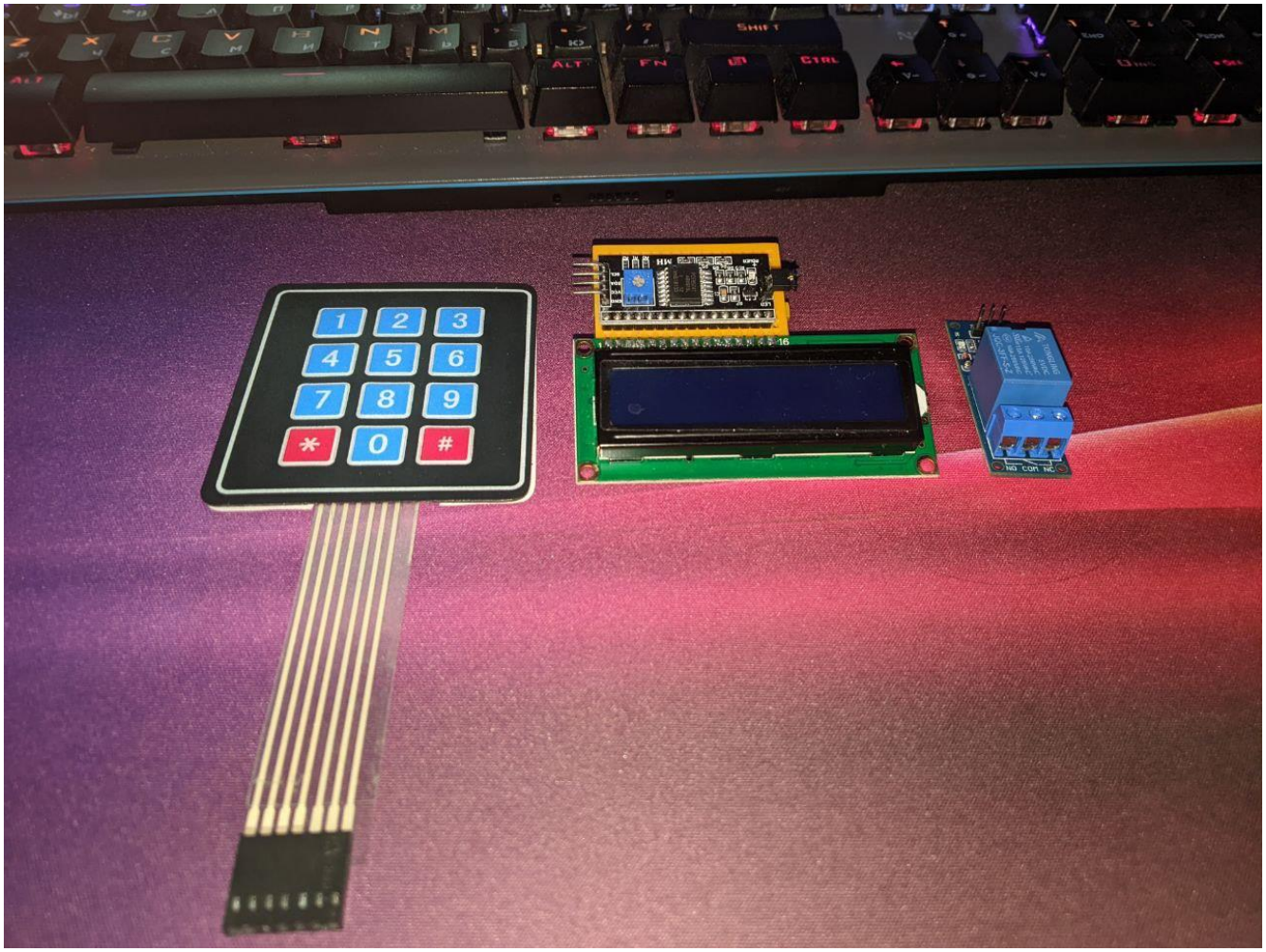

Рисунок 2.3 – Клавіатура та дисплей основного пристрою

Для сирени, в якості засобів індикації, було вибрано низьковольтний модуль буззера та, з перспективою заміни на адресні світлодіоди WS2812, вбудований в плату розробника світлодіод (рис. 2.4).

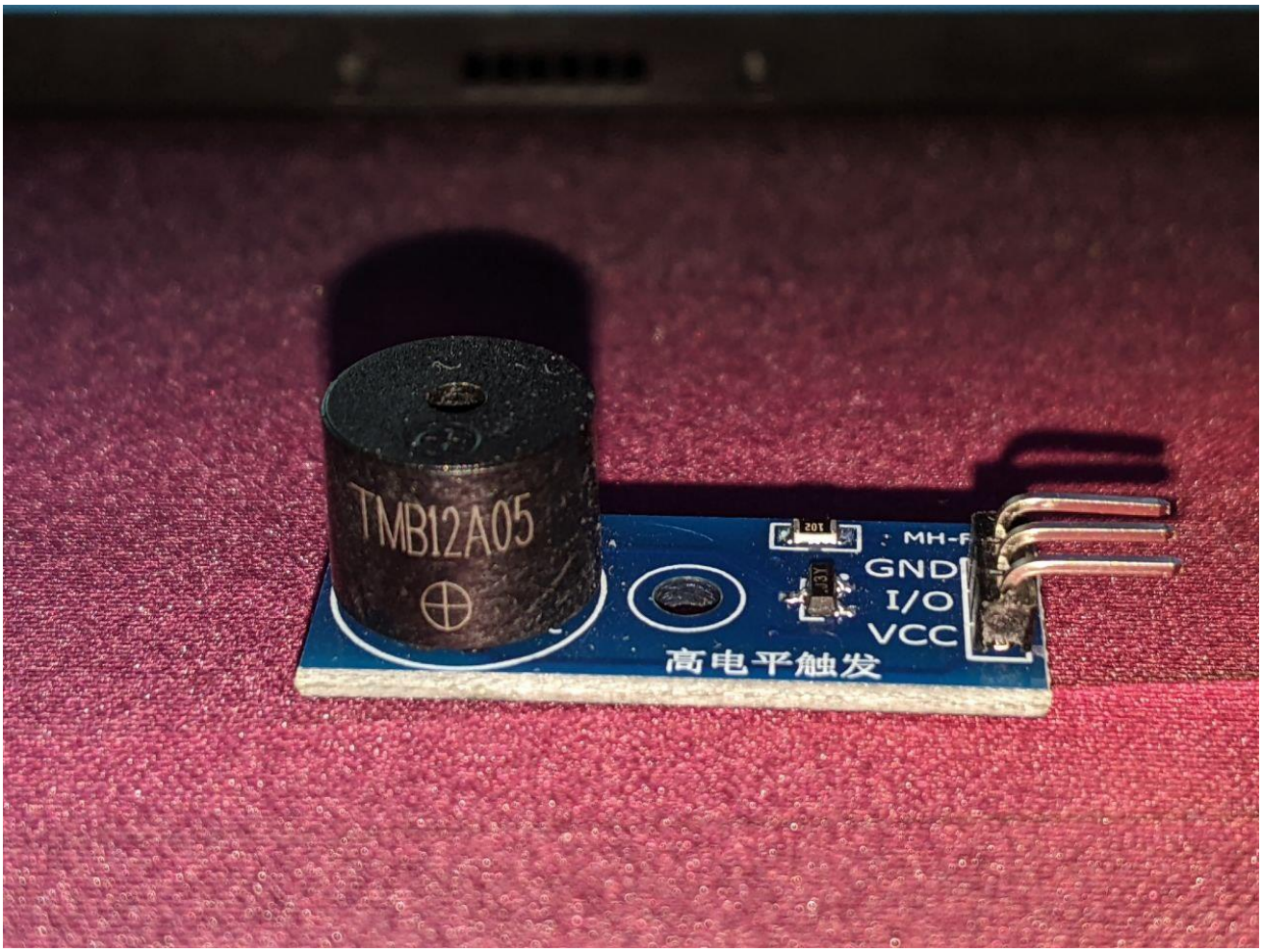

Рисунок 2.4 – Модуль буззера

2.2.2 Схема підключення

Розібравшись з платформою та іншими апаратними питаннями – можемо створити схему підключення усіх компонентів системи.

Керування системою та введення, власне, коду доступу буде здійснено через мембранну клавіатуру, що підключена трьома виводами до GPIO виходів та чотирма іншими виводами до GPIO EXTI входів головного мікроконтролера. Вивід зображення на екран здійснюється через I2C протокол і потребує виходу даних та тактування шини – SDA і SCL відповідно. Спілкування з сиреною реалізовано через USART протокол шляхом з’єднання її RX піна з TX піном кейпаду (закладена можливість повнодуплексного двостороннього спілкування, але у фінальній версії дані передаються виключно по шляху Кейпад->Сирена). Також, на кейпаді присутні два світлодіода, червоний та зелений, що виступають однозначним та зрозумілим показником статусу охорони системи. Для керування замком виведено окремий

GPIO_OUT LOCK_IO пін, який і даватиме команду логічною одиницею чи нулем зовнішньому пристрою на відкриття чи закриття замку (при зникненні живлення замок залишатиметься закритим, для потрапляння на об’єкт потрібно буде використати фізичний ключ).

Для сирени все просто – до МК підключений бузер, що керується ШИМ сигналом (для генерації частотного циклу різної гучності) та два світлодіоди для індикації (рис 2.5):

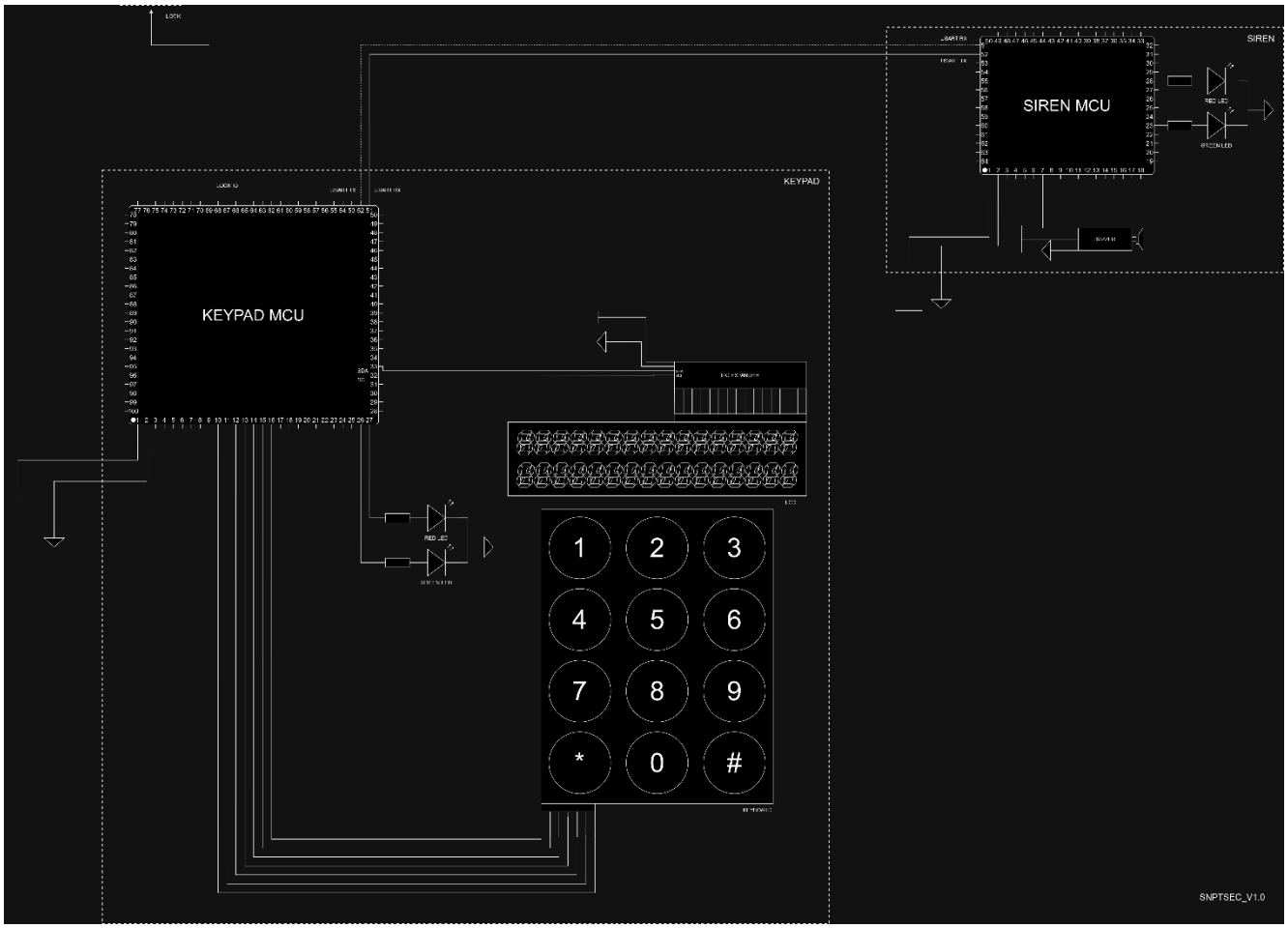

Рисунок 2.5 – Схема підключення розроблюваної системи контрою доступу

2.3 Розробка ПЗ та збірка прототипу

Спершу – створимо логіку обробки натиснутої клавіші клавіатури. Підключення відбуватиметься по схемі, описаній в пункті 2.1, тобто «три піна на вихід, чотири піна на вхід по перериванню».

Нижче (листінг 2.1) продемонстрований один з обробників переривань піна клавіатури, що підключена до піна мікроконтролера:

Лістинг 2.1 – Код обробника переривань піна клавіатури

```
void EXTI0_IRQHandler(void)
{

  HAL_GPIO_EXTI_IRQHandler(KEY_0_Pin);

  if(sec_it)
  {
      sec_it = false;
      return;
  }

  if(keypad.keyPending)
  {
      return;
  }

  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
     keypad.key = KEY_1;
     keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
      keypad.key = KEY_2;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
      keypad.key = KEY_3;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);

}
```

В частині лістингу коду вище показано обробку першого (нульового) EXTI входу (що налаштований на Rising Edge – фронт сигналу) з клавіатури – відбувається перемикання виходів МК до моменту, коли буде знайдено колонку, в якій знаходиться натиснута кнопка.

Змінна `sec_it` відповідає за зупинку обробки другого переривання, що неминуче виникне в момент повернення логічної одиниці на перевірений вихід піна.

Змінні `key` та `keyPending` структури keypad відповідають за передачу програмі натиснутої кнопки (кожна з яких представляє відповідний ASCII код символу) та ознаки наявності необробленої передачі цієї самої кнопки відповідно (лістинг 2.2).

Лістинг 2.2 – Код переліку доступних клавіш та структури їх обробника клавіатури

```
typedef enum
{
    KEY_NONE = 0x00,
    KEY_1    = 0x31,
    KEY_2    = 0x32,
    KEY_3    = 0x33,
    KEY_4    = 0x34,
    KEY_5    = 0x35,
    KEY_6    = 0x36,
    KEY_7    = 0x37,
    KEY_8    = 0x38,
    KEY_9    = 0x39,
    KEY_ZIR  = 0x2A,
    KEY_0    = 0x30,
    KEY_RESH = 0x23
}key_pressed;

typedef struct
{
    key_pressed key;
    bool keyPending;
}keypad_t;
```

В основній програмі, після ініціалізації усієї периферії, обробка відбувається в нескінченному while циклі, з затримкою в 100мс між проходами (також присутня затримка в 3000мс, що встановлюється програмою після виконання команд встановлення/зняття з охорони, а також при вводі невірного коду доступу) (лістинг 2.3).

Лістинг 2.3 – Код основного нескінченного циклу

```
while (1)
{
 HAL_Delay(100); // 100ms pause
 if(IsProcessingPaused())
 {
  static uint8_t pause;
  pause++;
  if(pause >= 30) // 3 sec pause
  {
      ResetPauseFlag();
      keypad.keyPending = false;
      pause = 0;
      SendDisplayMgs(LCD_DEFAULT);
  }
 }
 else
 {
  if(keypad.keyPending)
  {
      ProcessKey(keypad.key); // main processing func
      keypad.keyPending = false;
  }
 }
}
```

Функція `ProcessKey` викликається при виконанні умови «є необроблена клавіша» і являє собою звичайний switch, що оброблює клавіші на основі поточного стану системи та віддає відповідні команди периферії до виконання (лістинг 2.4):

Лістинг 2.4 – Код функції обробки натиснутої клавіші клавіатури

```
void ProcessKey(key_pressed key)
{
#define MAX_PASS_LENGTH 10
    static uint8_t passLength = 0;
    KeyBeep();
    switch(sysState)
    {
    case SYS_DISARMED:
       if(key == KEY_RESH)
       {
          sysState = SYS_ARMED;
          HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(LD5_GPIO_Port, LD5_Pin, GPIO_PIN_SET);
          DoorsCmd(DOORS_CLOSE);
          SendSirenMsg(SIREN_ARM);
```

```
            SendDisplayMgs(LCD_ARM);
            SetPauseFlag();
        }
        break;
    case SYS_ARMED:
        if((key != KEY_ZIR) && (key != KEY_RESH))
        {
            if(passLength < MAX_PASS_LENGTH)
            {
                passLength++;
                char newSym = key;
                strncat(passStr, &newSym, 1);
                SendDisplayMgs(LCD_NEWCHAR);
            }
            else
            {
                do{}while(0);
            }
        }
        else
        {
            if(key == KEY_RESH)
            {
                if(!strcmp(passStr, allowStr))
                {
                    sysState = SYS_DISARMED;
                    HAL_GPIO_WritePin(LD5_GPIO_Port, LD5_Pin, GPIO_PIN_RESET);
                    HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);
                    DoorsCmd(DOORS_OPEN);
                    SendDisplayMgs(LCD_DISARM);
                    SendSirenMsg(SIREN_DISARM);
                    SetPauseFlag();
                }
                else if(strlen(passStr))
                {
                    SendDisplayMgs(LCD_WRONG_PASS);
                    SendSirenMsg(SIREN_WRONG_PASS);
                    SetPauseFlag();
                }

            }
            else if(key == KEY_ZIR)
            {
                if(strlen(passStr))
                    SendDisplayMgs(LCD_DEFAULT);
            }
            memset(passStr,0,strlen(passStr));
            passLength = 0;
        }
        break;
    }
}
```

Повний лістинг коду доступний в розділі Додаток А.

2.4 Тестування системи

Після збірки системи спершу перевіримо коректність роботи клавіатури та логіки обробки клавіш. Для цього підключимо дебагер до основної плати та запустимо відображення змінної клавіш в реальному часі (рис. 2.6):

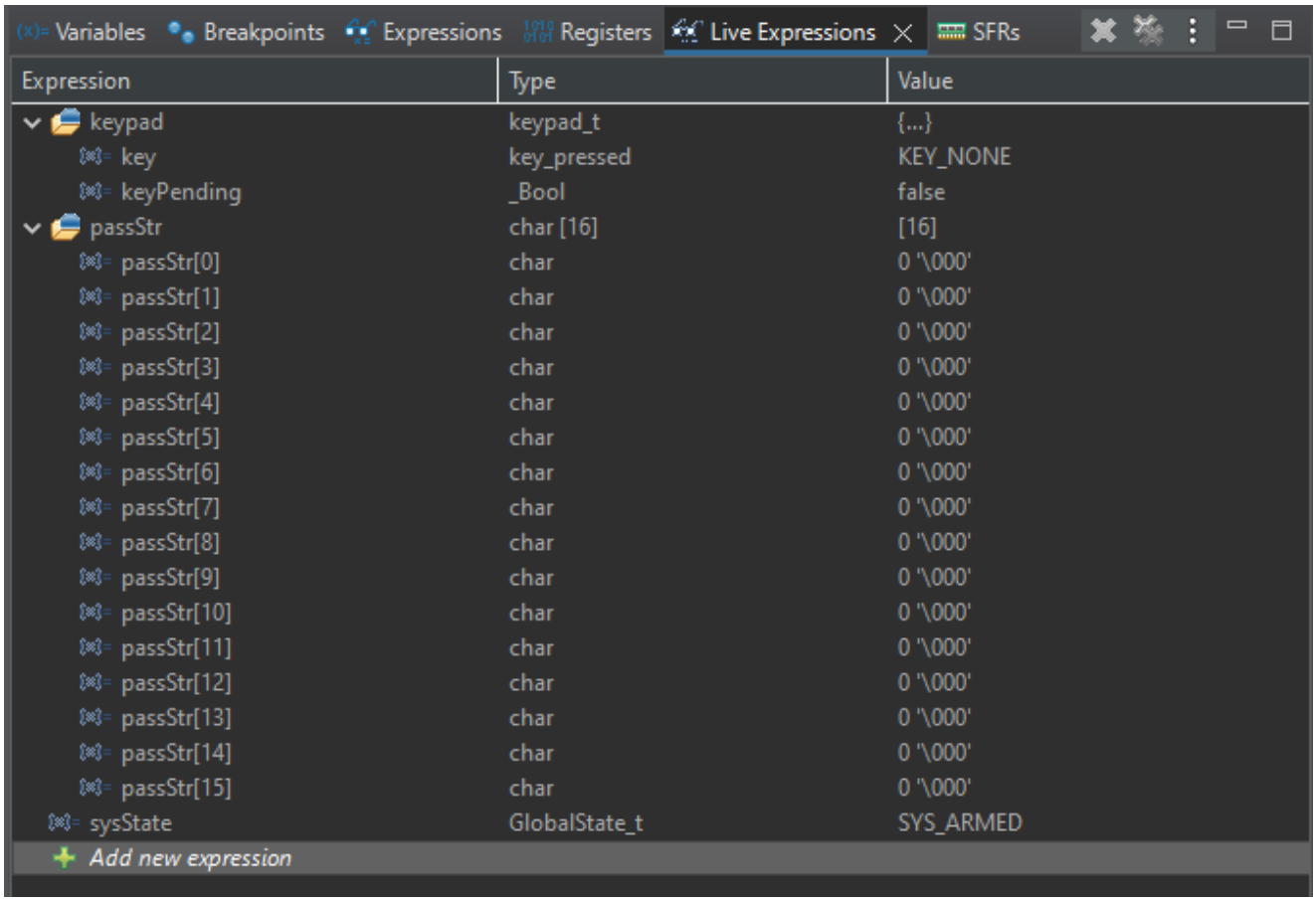

Рисунок 2.6 – Початковий стан внутрішнії змінних системи

Перевіримо кожну з нумерних клавіш (рис. 2.7):

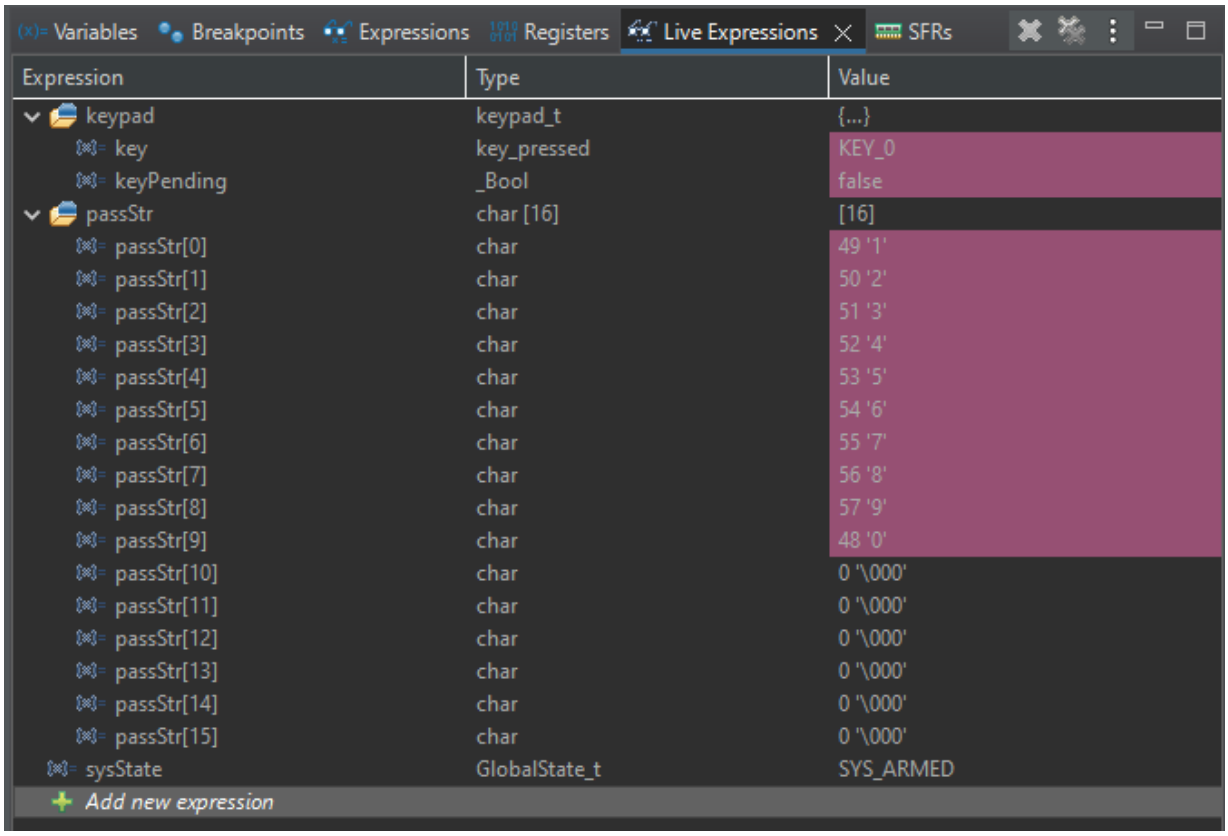

Рисунок 2.7 – Перевірка правильності прийому натиснутих клавіш

Спроба очистки масива введених символів клавішою «*» відображена на рис. 2.8:

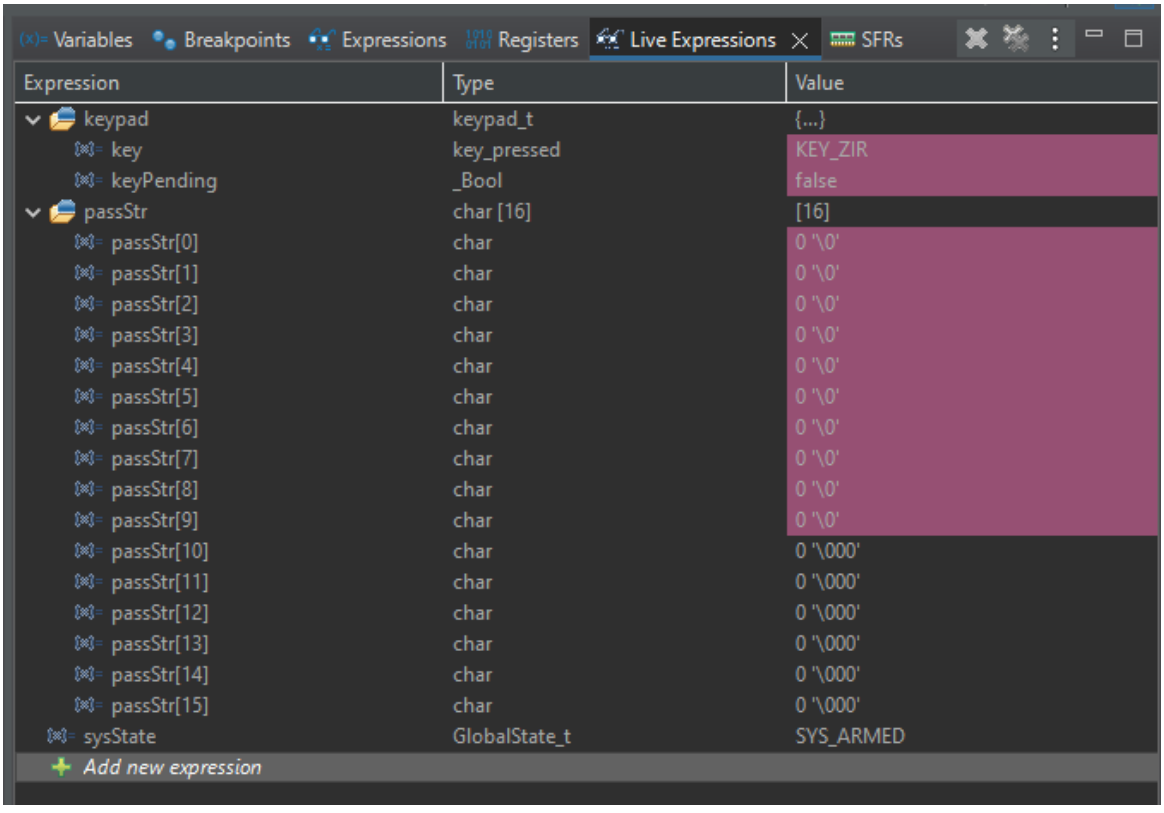

Рисунок 2.8 – Успішне очищення масиву символів

Введемо правильний пароль (рис 2.9):

Variables | Breakpoints | Expressions | Registers | Live Expressions | SFRs

Expression	Type	Value
keypad	keypad_t	{...}
key	key_pressed	KEY_5
keyPending	_Bool	false
passStr	char [16]	[16]
passStr[0]	char	54 '6'
passStr[1]	char	54 '6'
passStr[2]	char	53 '5'
passStr[3]	char	0 '\0'
passStr[4]	char	0 '\0'
passStr[5]	char	0 '\0'
passStr[6]	char	0 '\0'
passStr[7]	char	0 '\0'
passStr[8]	char	0 '\0'
passStr[9]	char	0 '\0'
passStr[10]	char	0 '\000'
passStr[11]	char	0 '\000'
passStr[12]	char	0 '\000'
passStr[13]	char	0 '\000'
passStr[14]	char	0 '\000'
passStr[15]	char	0 '\000'
sysState	GlobalState_t	SYS_ARMED
allowStr	char [16]	[16]
allowStr[0]	char	54 '6'
allowStr[1]	char	54 '6'
allowStr[2]	char	53 '5'
allowStr[3]	char	0 '\000'
allowStr[4]	char	0 '\000'
allowStr[5]	char	0 '\000'
allowStr[6]	char	0 '\000'
allowStr[7]	char	0 '\000'
allowStr[8]	char	0 '\000'
allowStr[9]	char	0 '\000'
allowStr[10]	char	0 '\000'
allowStr[11]	char	0 '\000'
allowStr[12]	char	0 '\000'
allowStr[13]	char	0 '\000'
allowStr[14]	char	0 '\000'
allowStr[15]	char	0 '\000'
Add new expression		

Рисунок 2.9 – Введення правильного коду доступу, до натискання на «#»

Натиснемо клавішу «#» для початку обробки введеного паролю (рис 2.10):

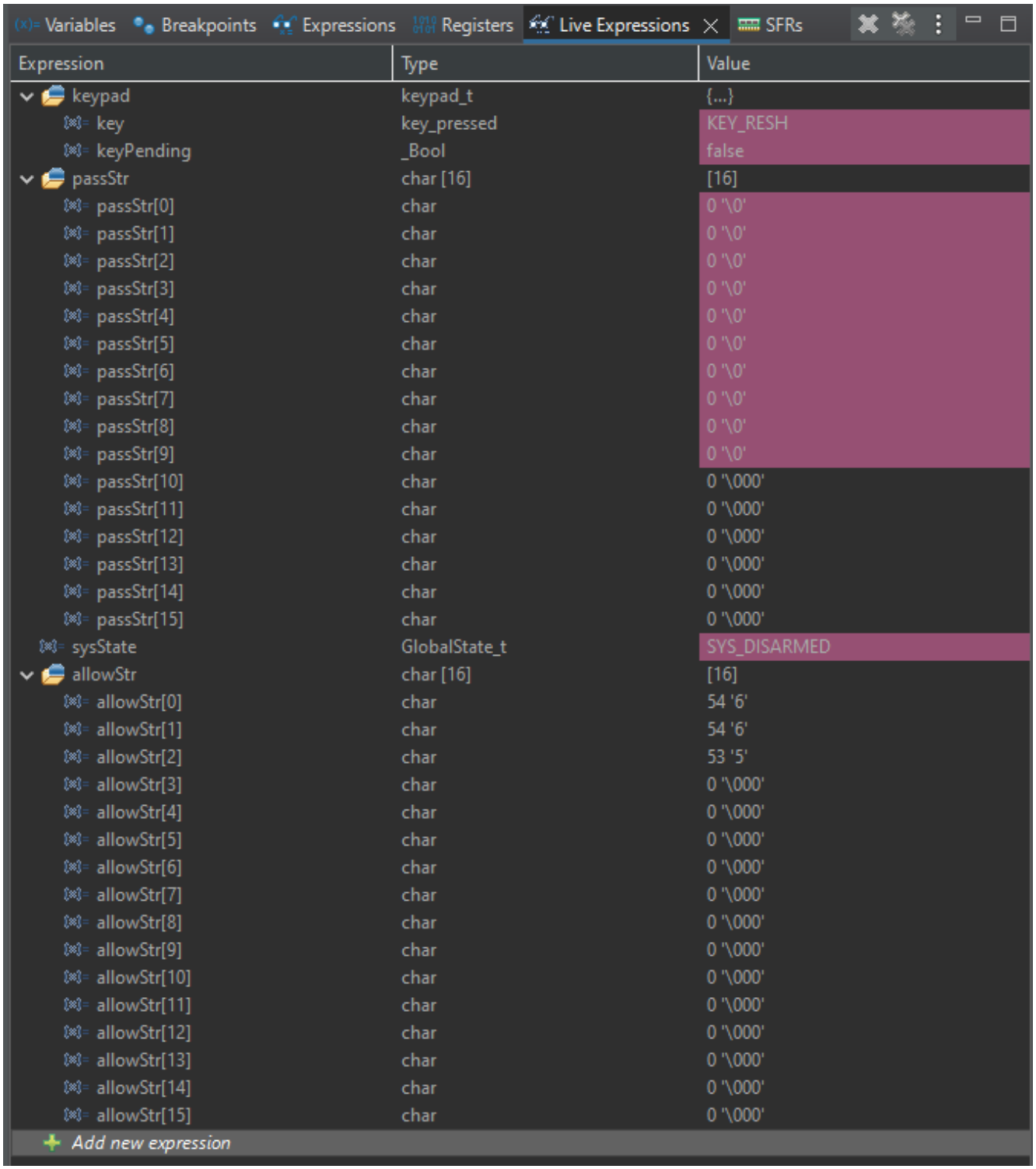

Рисунок 2.10 – Введення правильного коду доступу, після натискання на «#»

Як бачимо – введені з клавіатури символи опрацьовуються згідно з ER, повністю покриваючи наявні вимоги.

Вигляд системи після повної збірки відображено на рисунку 2.11:

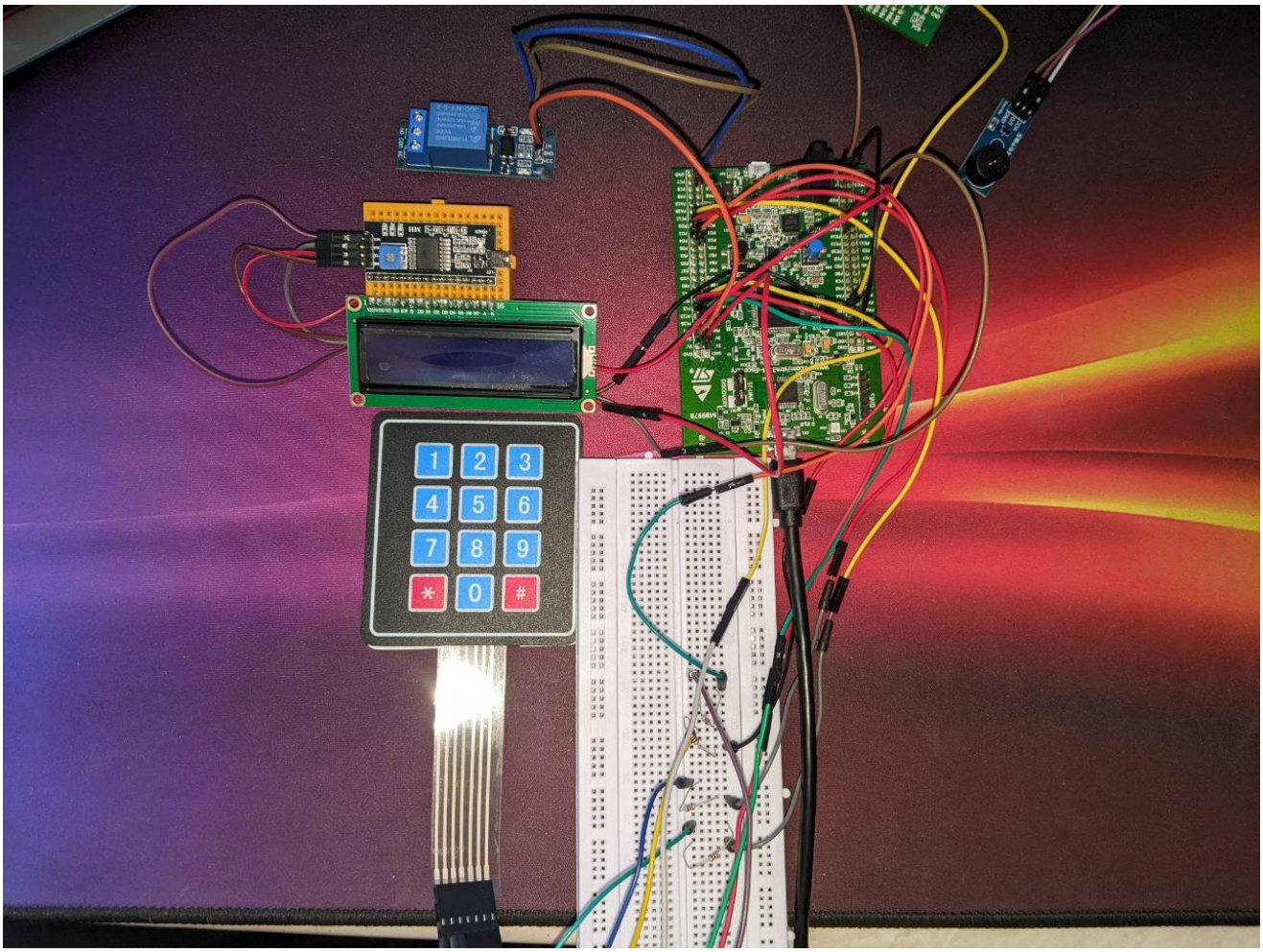

Рисунок 2.11 – Система в зборі

Подамо живлення на плату та протестуємо її в різних сценаріях (рис 2.12, рис. 2.13, рис. 2.14, рис. 2.15, рис. 2.16, рис. 2.17):

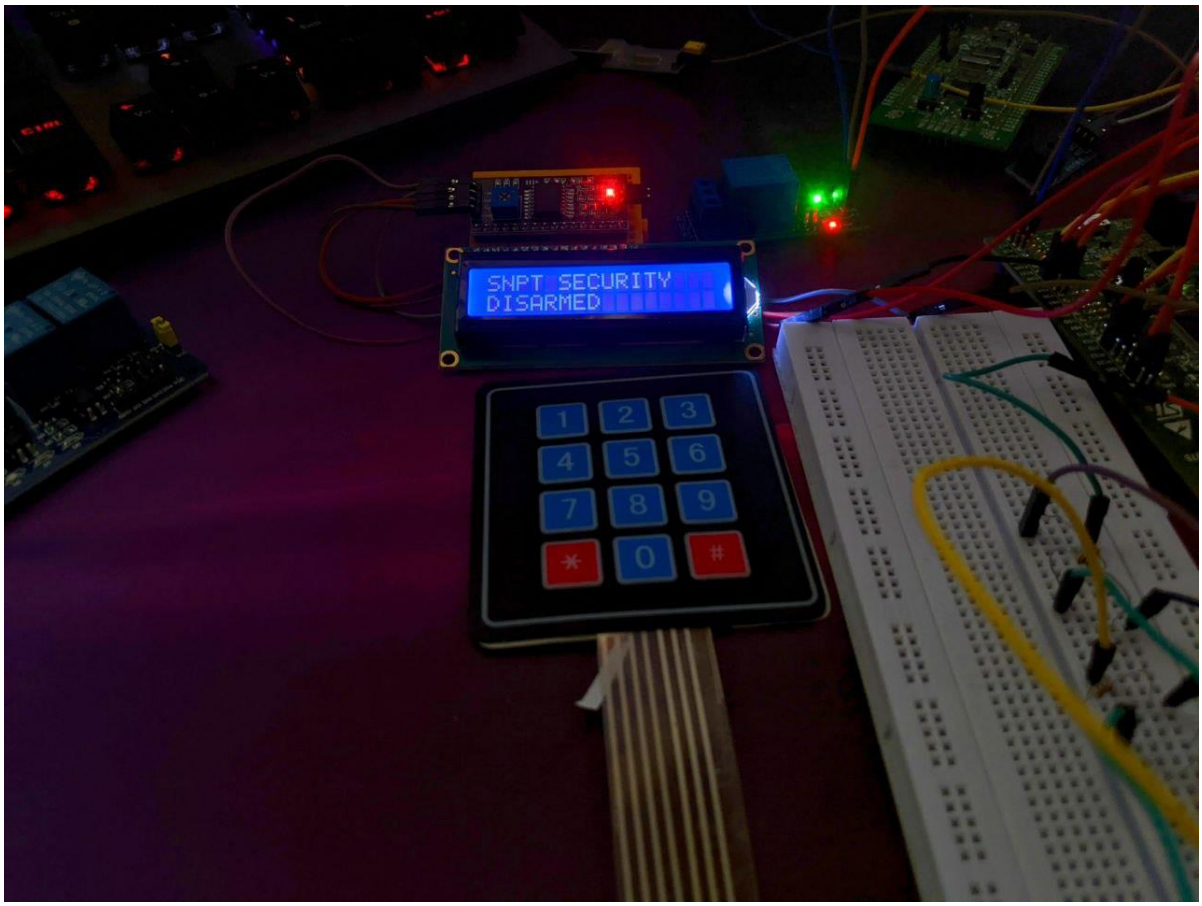

Рисунок 2.12 – Система не під охороною, двері відкриті

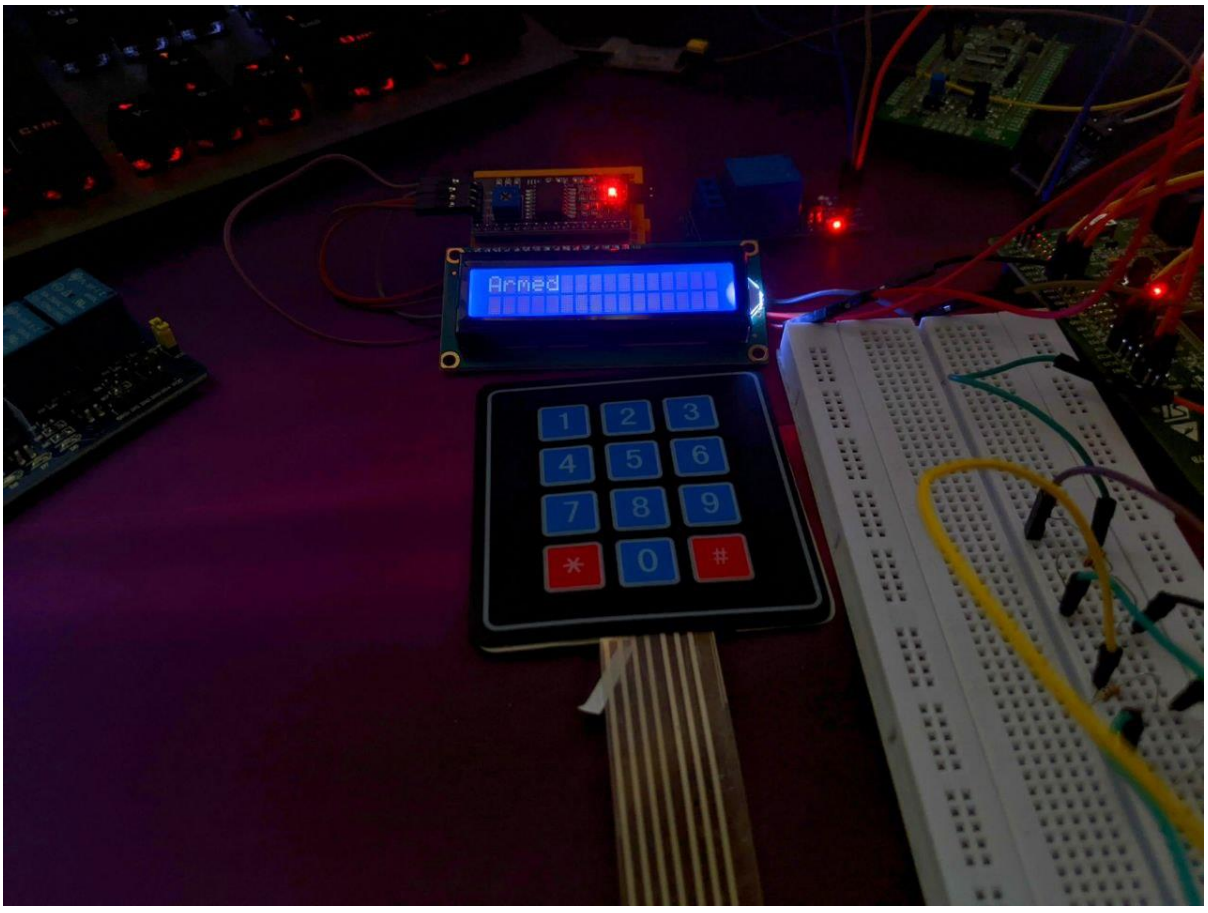

Рисунок 2.13 – Система встановлена під охорону, двері закриті

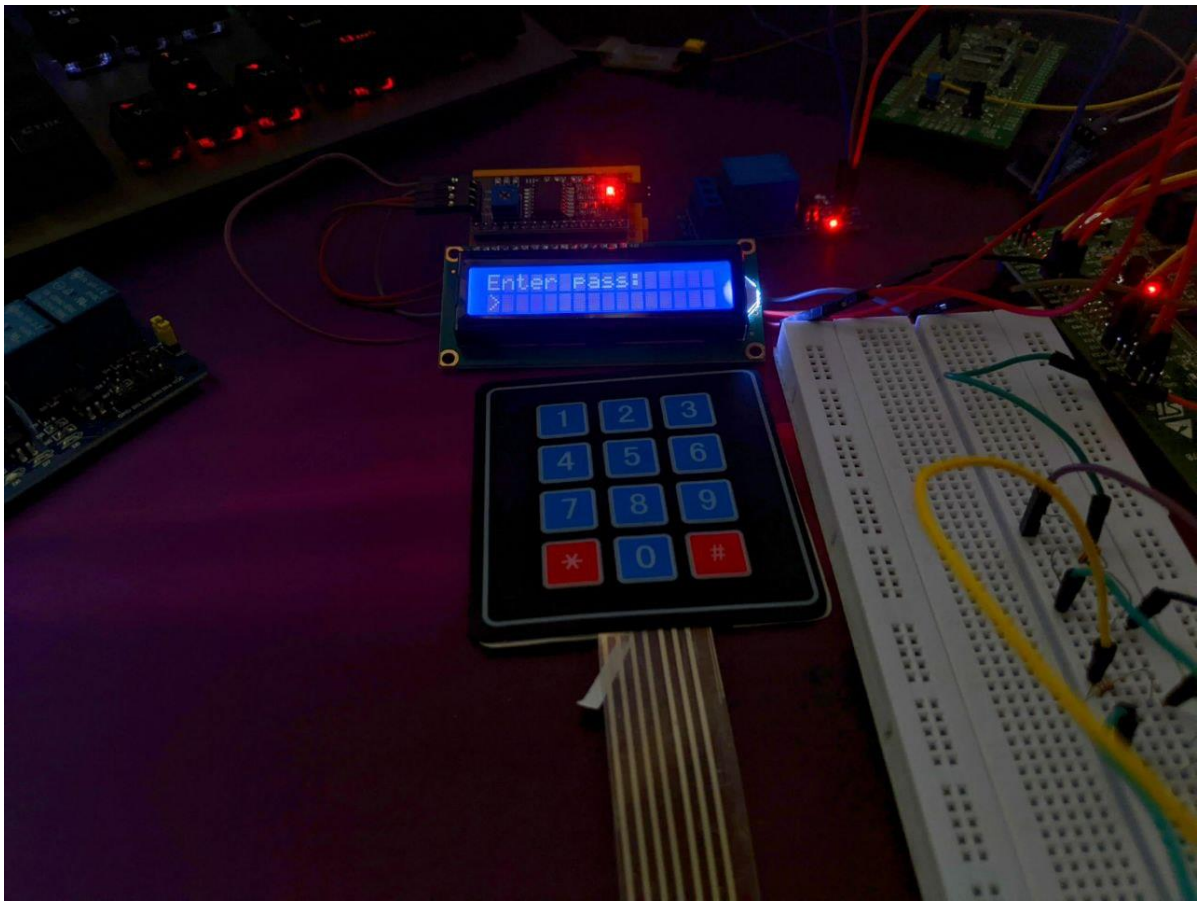

Рисунок 2.14 – Поле вводу коду доступу

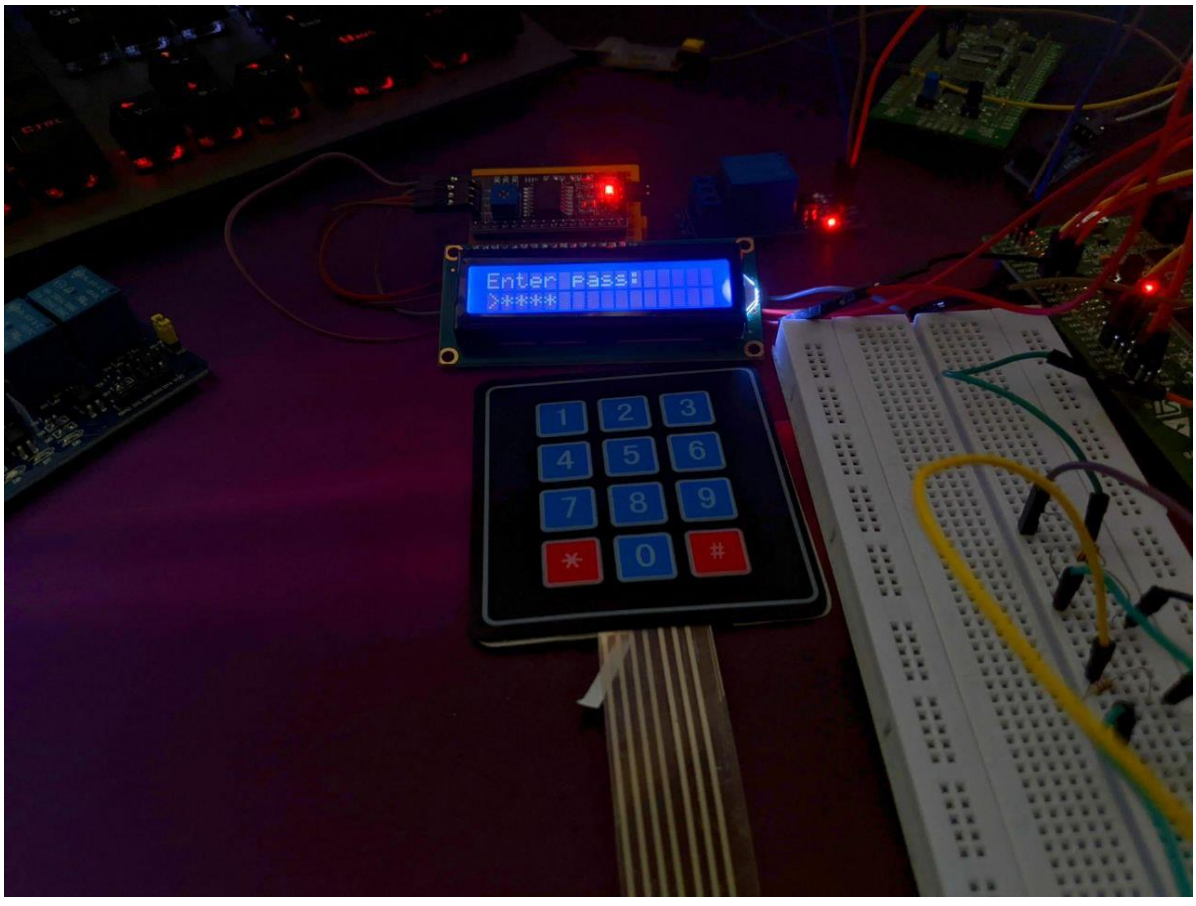

Рисунок 2.15 – Приховане відображення введеного коду доступу

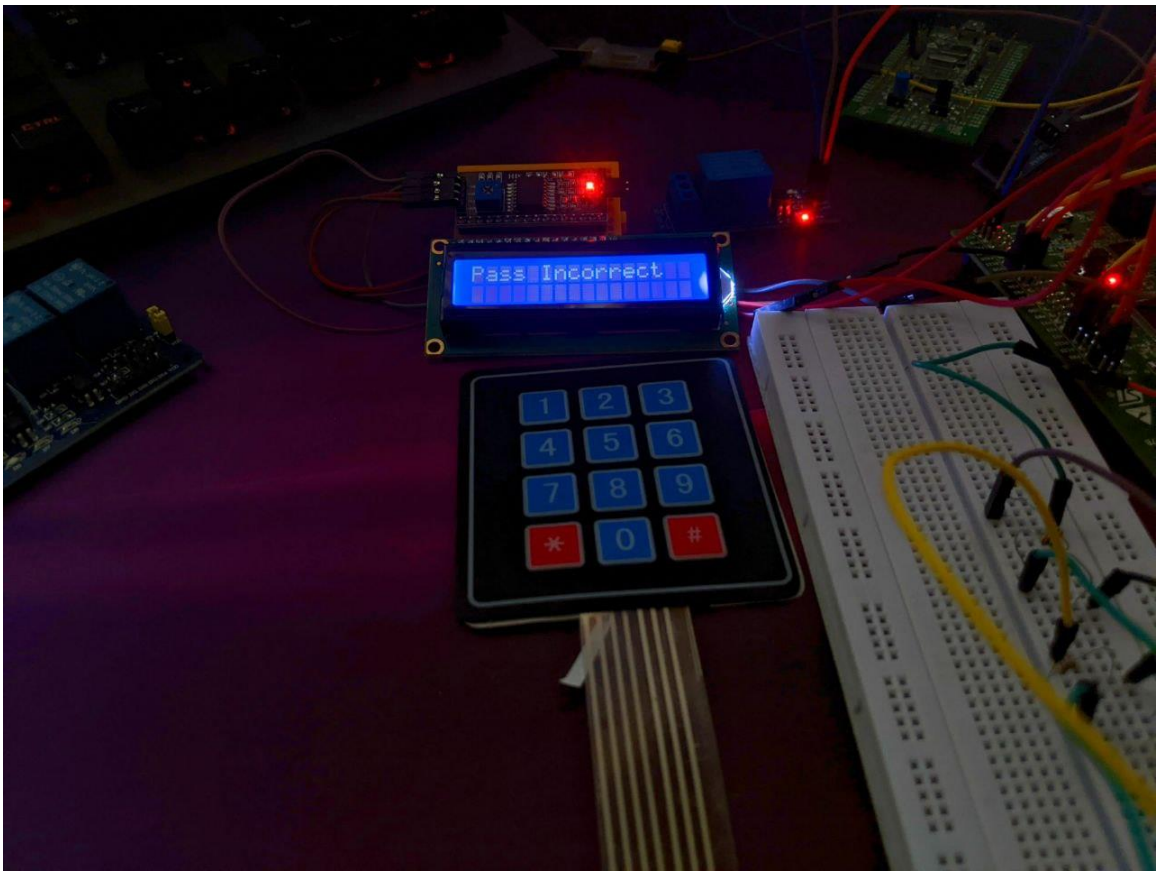

Рисунок 2.16 – Повідомлення про невдале введення коду доступу

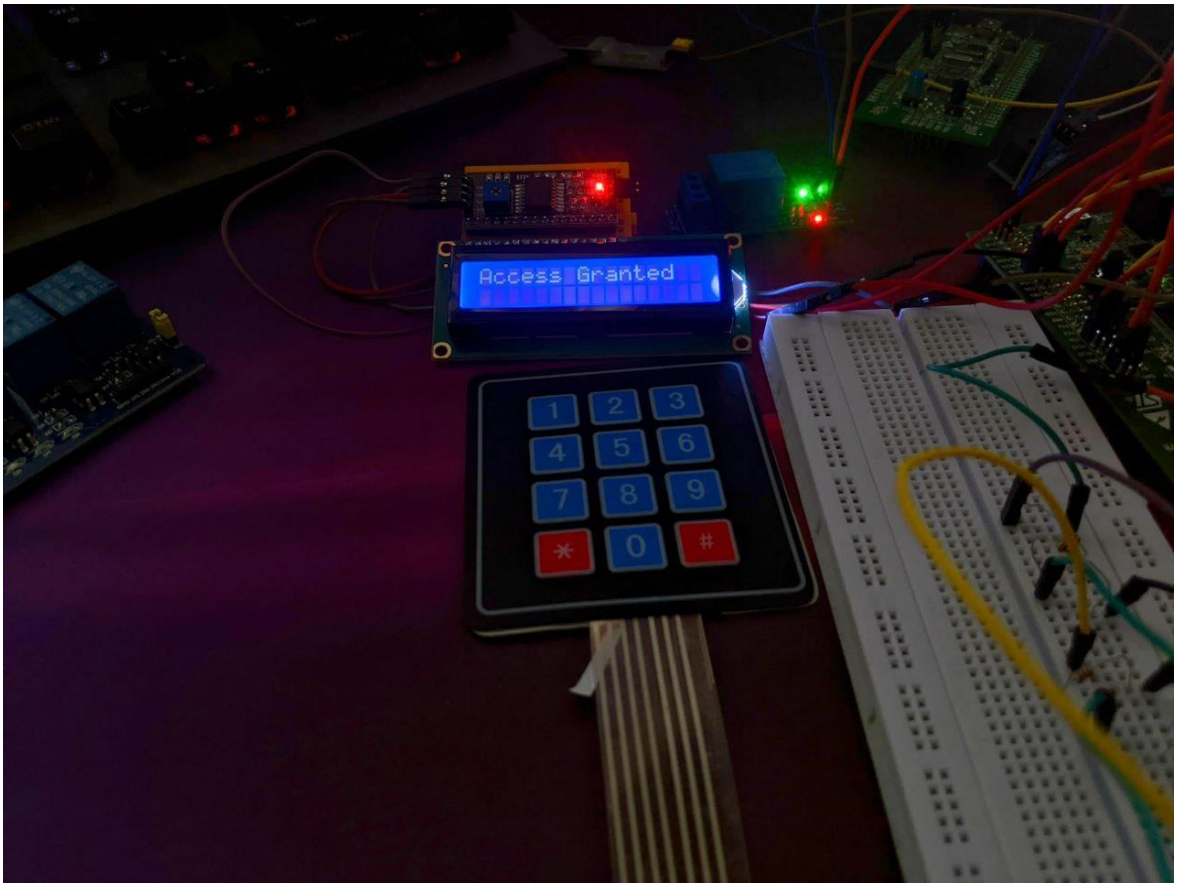

Рисунок 2.17 – Введений правильний код доступу, система знята з охорони, двері відкриті

3 АНАЛІЗ ВРАЗЛИВОСТЕЙ ОХОРОННОЇ СИСТЕМИ ДОСТУПУ

Уявімо ситуацію – в компанію по розробці та супроводженню вбудованих пристроїв приходить лист. В листі вказано, що одна фірма, по рекомендації нового менеджера, замовила у іншої фірми розробку системи доступу для охорони свого об’єкту «новітніми технологіями».

Спеціалісти фірми розробника встановили та підключили систему, розповіли про принцип роботи та дали можливість налаштувати пароль відповідальній за об’єкт людині.

Через кілька тижнів об’єкт, на якому була встановлена система, було пограбовано, загальна шкода склала більше квартального доходу усієї компанії. Слідів зламу виявлено не було.

По результатам внутрішнього розслідування людина, що була відповідальна за об’єкт і що єдина знала код доступу до об’єкту виявилась непричетною до інциденту. Також, за запитом до фірми розробника, було отримано відповідь про неможливість компрометації їх системи безпеки та відмову в подальшій співпраці.

Вам, як незалежній стороні, доручено виявити усі можливі загрози безпеці розробленій системі доступу для подальших претензійних заяв у бік фірми-розробника.

3.1 Аналіз підключення

Перше, що варто зробити після приїзду на об’єкт – це аналіз підключення системи. Після вивчення усіх підключень (без розбору пристрою) було складено наступну діаграму (рис. 3.1):

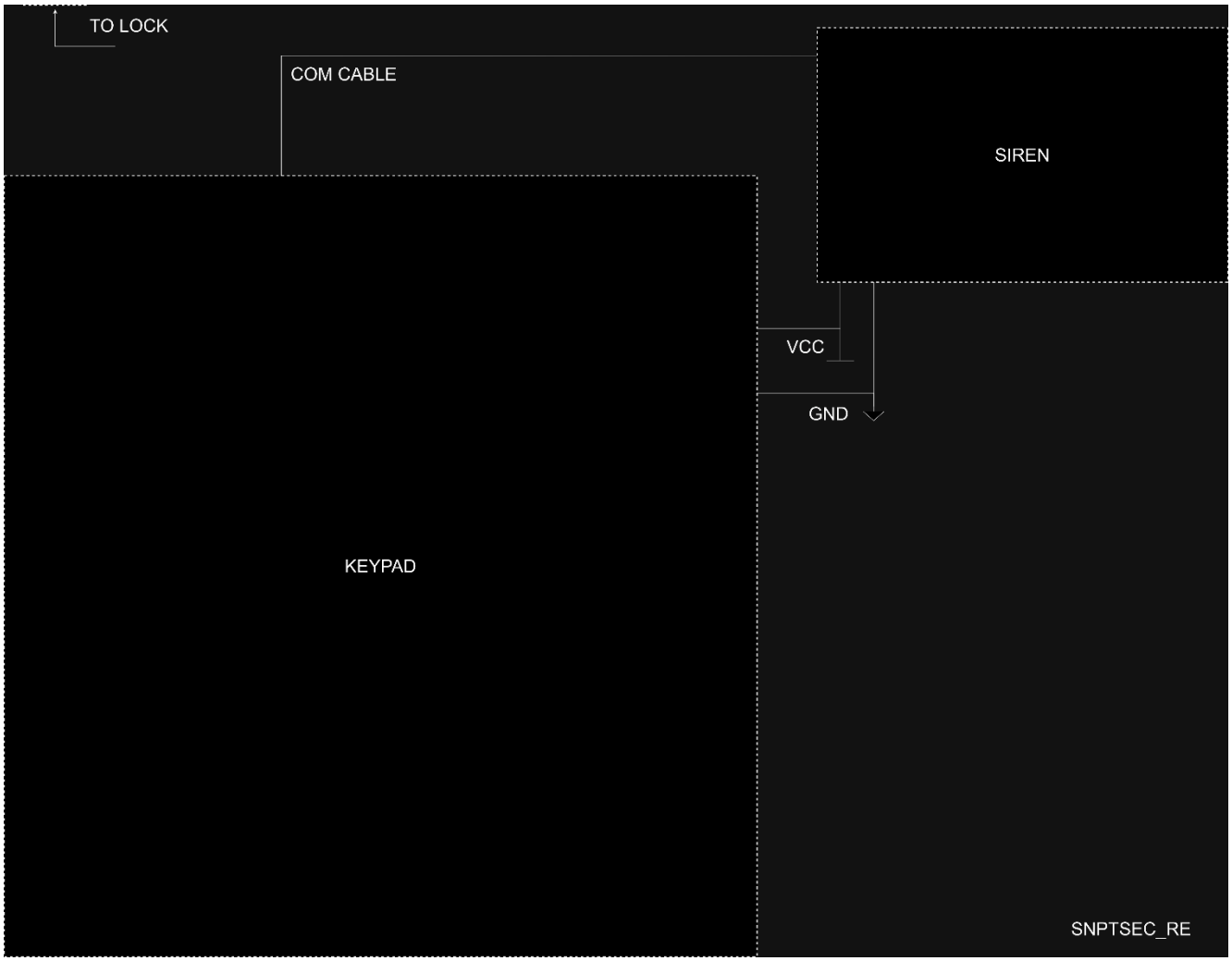

Рисунок 3.1 – Діаграма зовнішніх підключень системи

Як можна побачити – найслабшими місцями системи є саме місця з’єднання трьох пристроїв – кейпаду, сирени та замка. Щоб підтвердити попередні висновки щодо можливого отримання зловмисником доступу до одного з сигнальних кабелів – потрібно провести аналіз поведінки пристрою.

3.2 Аналіз програмного забезпечення

Для цього, використовуючи офіційне ПЗ від виробника мікроконтролерів ST – «STVP» (ST Visual Programmer) спробуємо зчитати прошивку з головного керуючого пристрою. Ця задача може виявитись неможливою, адже для блокування зчитування даних з мікроконтролера розробникам потрібно всього лиш встановити налаштування ReadoutProtection (RDP) хоча б на другий з трьох рівнів, не лишаючи можливості третім особам отримати доступ до байтів прошивки без повного її знищення. Але, зважаючи на загальну недбалість в забезпеченні захисту системи

від вторгнення, висновок по якому було зроблено ще на стадії поверхневого вивчення системи, шанс на успішне зчитування прошивки лишається високим.

Отже, запустивши STVP нам потрібно вибрати програматор, за допомогою якого буде здійснено зчитування, режим програмування та сам МК, з якого буде проведено зчитування прошивки. Вигляд вікна налаштувань показаний на рис. 3.2

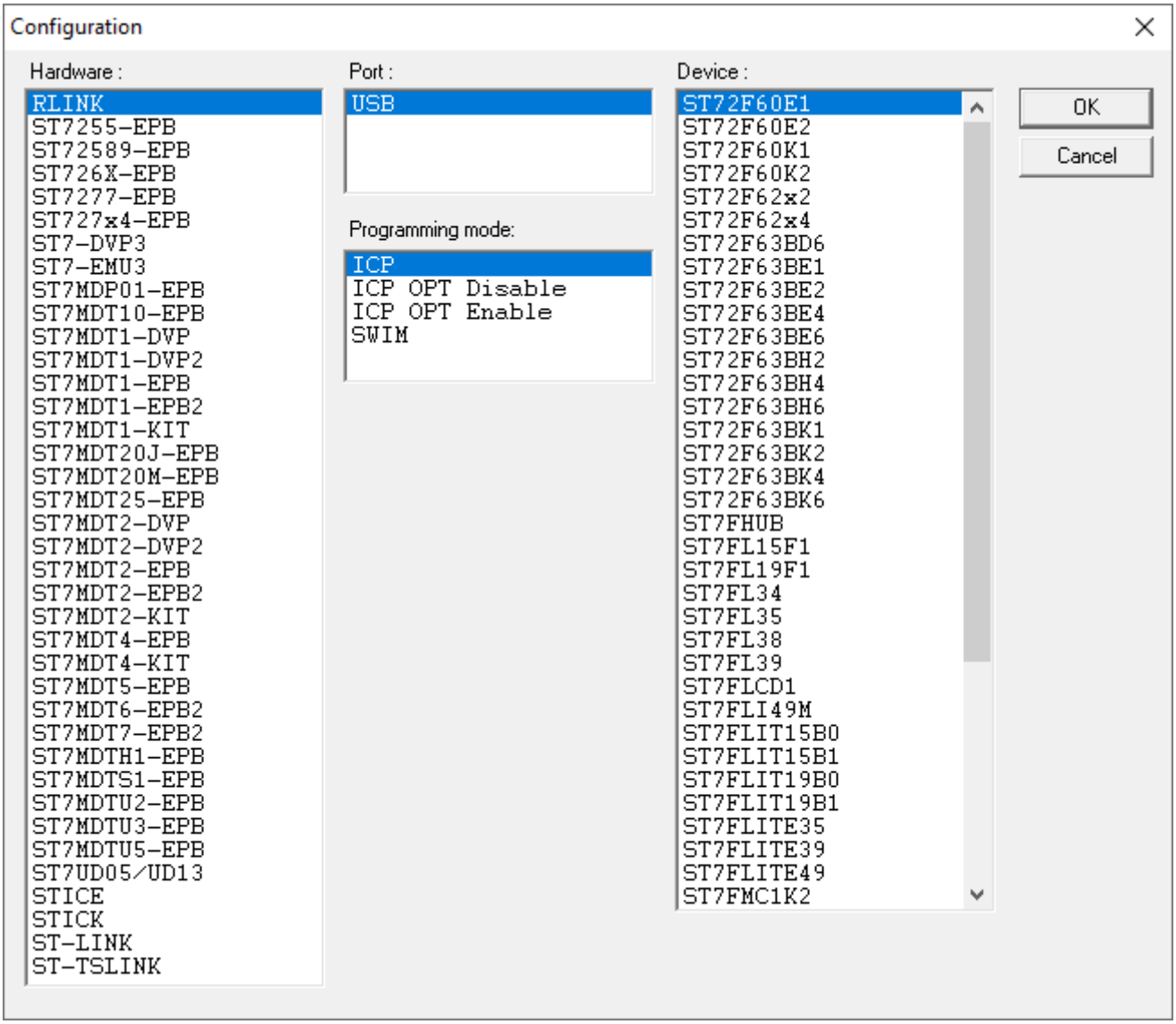

Рисунок 3.2 – Вікно конфігурації ST Visual Programmer

З першим пунктом для STM32 в цілому проблем бути не має – використаємо звичайний клон ST-LINK, який програмне забезпечення не зможе відрізнити від оригінального (рис. 3.3):

Рисунок 3.3 – Клон ST-LINK V2

З другим пунктом складніше, адже прошивка могла бути залита як через JTAG, так і через SWD (режим SWIM використовується тільки для 8-ми бітних мікроконтролерів STM8, тож його до уваги не беремо). На основі попереднього досвіду зробимо припущення, що для прошивки використовувався SerialDebugWire і відповідні піни все ще присутні всередині неї (у випадку відсутності сконфігурованих пінів SWD потрібно було б утримувати МК в режимі Reset до моменту повного підключення програматора).

Третій пункт найпростіший – нам усього лише варто поглянути на чип, що розпаяний на платі керуючого пристрою (рис. 3.4):

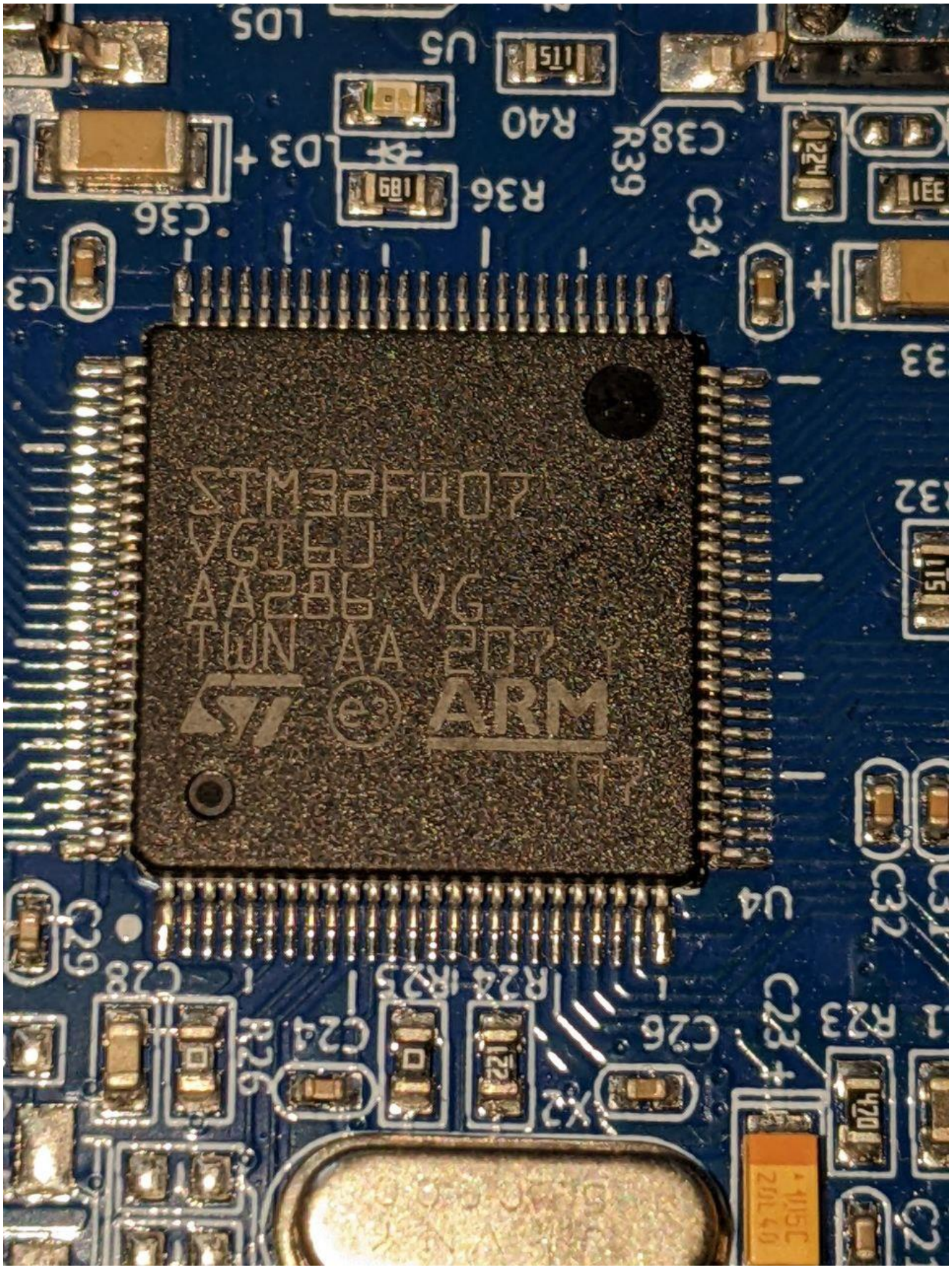

Рисунок 3.4 – Маркування на мікроконтролері кейпаду

Отже, кінцева конфігурація STVP виглядатиме наступним чином (рис. 3.5):

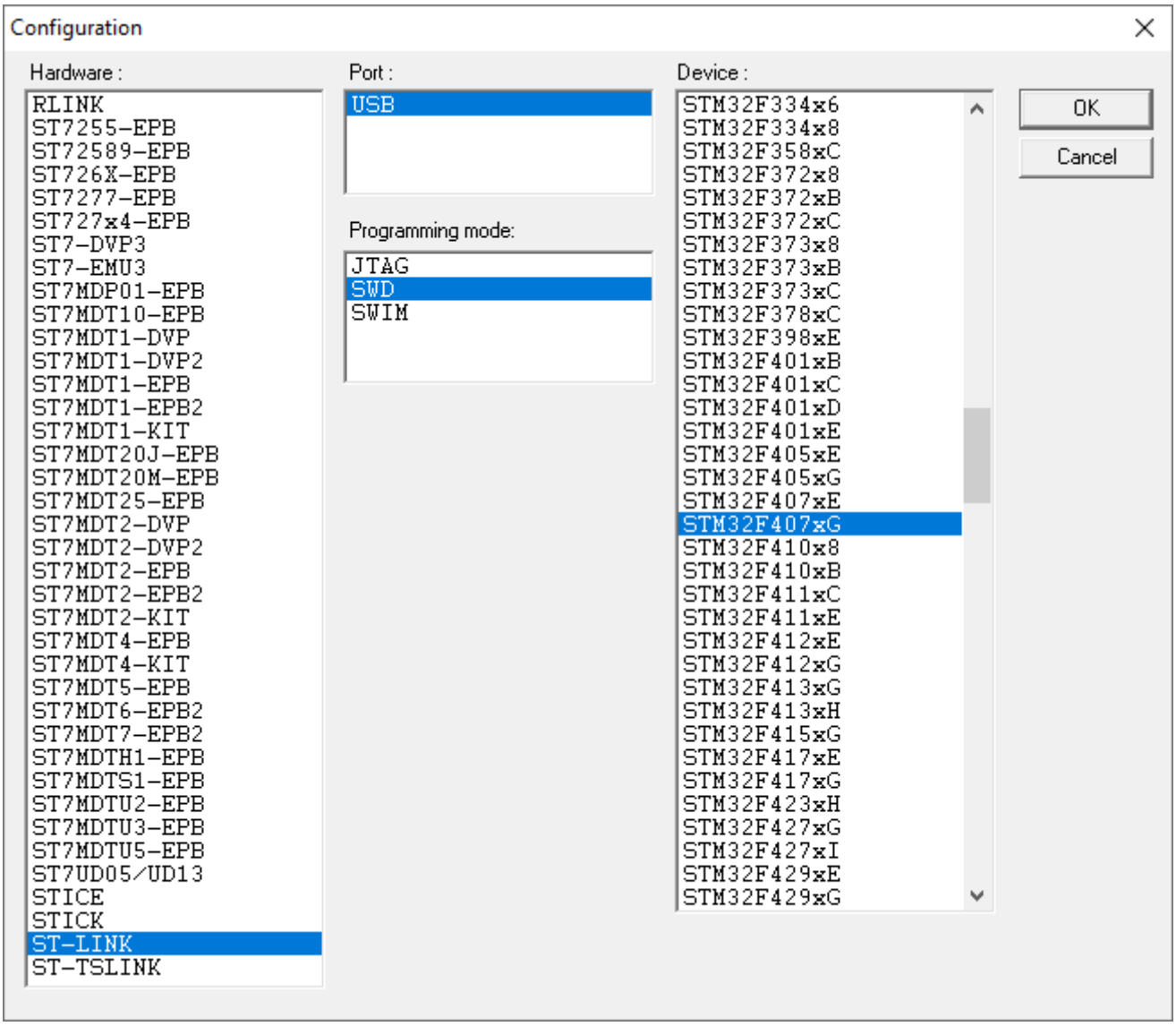

Рисунок 3.5 – Правильна конфігурація STVP для зчитування прошивки

Тепер, після натискання на клавішу «ОК», підключимо програматор до відповідних пінів мікроконтролера. Щоб знайти такі, скористаємося офіційним ПЗ STM32CubeMX та перевіримо стандартну розпіновку мікроконтролера при заданні SWD в якості режиму програмування. Налаштовуємо конфігурацію відповідно до рисунку 3.6:

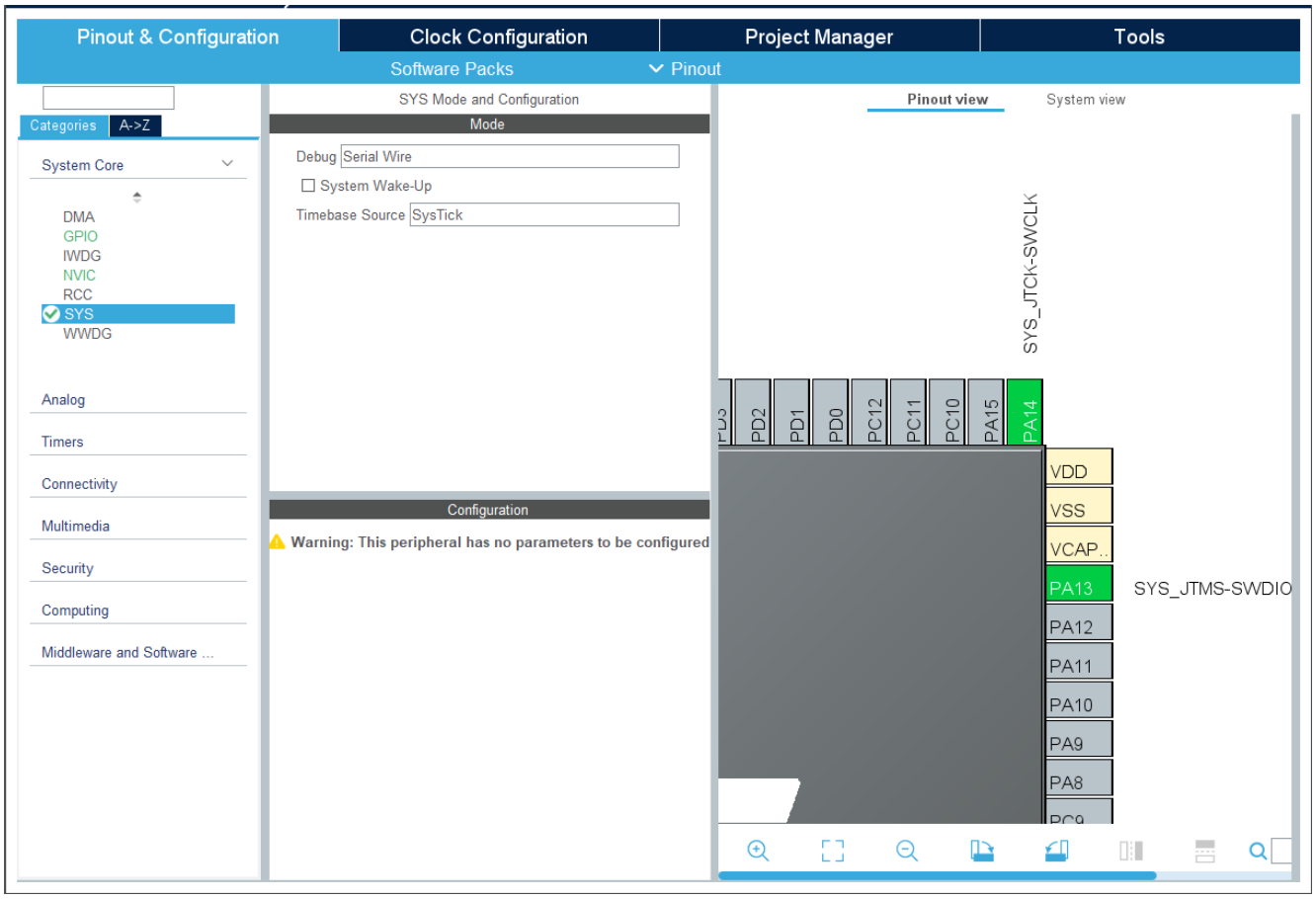

Рисунок 3.6 – Аналіз можливих пінів SWD в STM32CubeMX

Як ми можемо побачити – стандартними пінами SWD є PA13 і PA14, що відповідають за передачу даних (SWDIO) та тактування лінії (SWDCLK) відповідно.

Підключаємо програматор до пристрою з одного боку та до ПК з іншого і намагаємось зчитати прошивку (рис. 3.7):

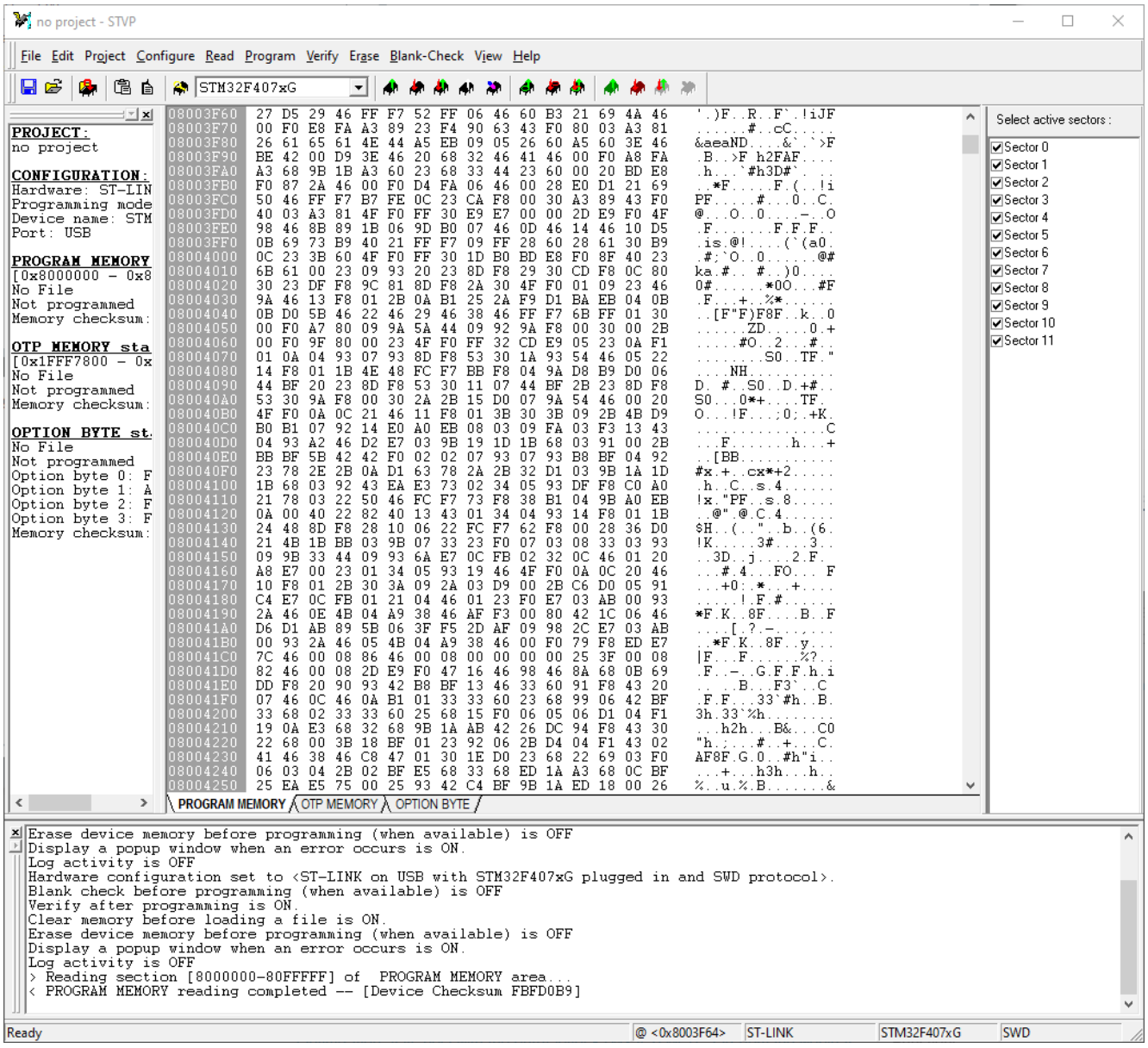

Рисунок 3.7 – hex дамп прошивки кейпаду

Захисту від зчитуванні на мікроконтролері не було, а це значить що hex файл прошивки тепер доступний для аналізу!

Витративши десять хвилин часу, ми натикаємося на підозрілу послідовність байтів, що показана на рисунку 3.8:

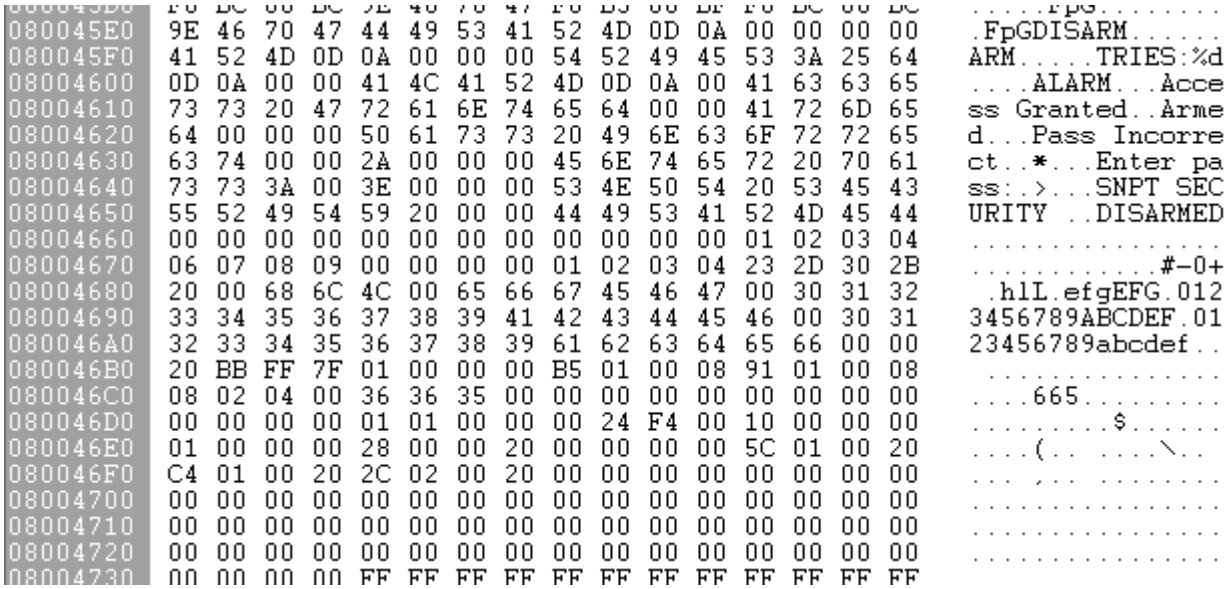

Рисунок 3.8 – Символьні рядки в пам’яті мікроконтролера

Здається, що послідовність байтів під питанням є місцем зберігання послідовності символів – символьних рядків. Особливо, з поміж іншого, виділяється рядок 080046Cx, в якому зберігається число «665». Запам’ятаємо цю послідовність символів, вона нам знадобиться в майбутньому.

3.3 Аналіз поведінки пристрою та взаємодії з зовнішніми пристроями

Після аналізу прошивки системи варто перевірити загальну захищеність прокотолів спілкування між пристроями. Раніше, при аналізі підключень, було виявлено дві сигнальні лінії з невідомим нам протоколом спілкування. Вважаючи, що знайдена послідовність символів є кодом доступу до системи – підключимо логічний аналізатор до обох ліній та спробуємо поставити та зняти систему з охорони.

Підключення відбуватиметься напряму, в розрив між головним та отримуючими пристроями, як показано на рис. 3.9:

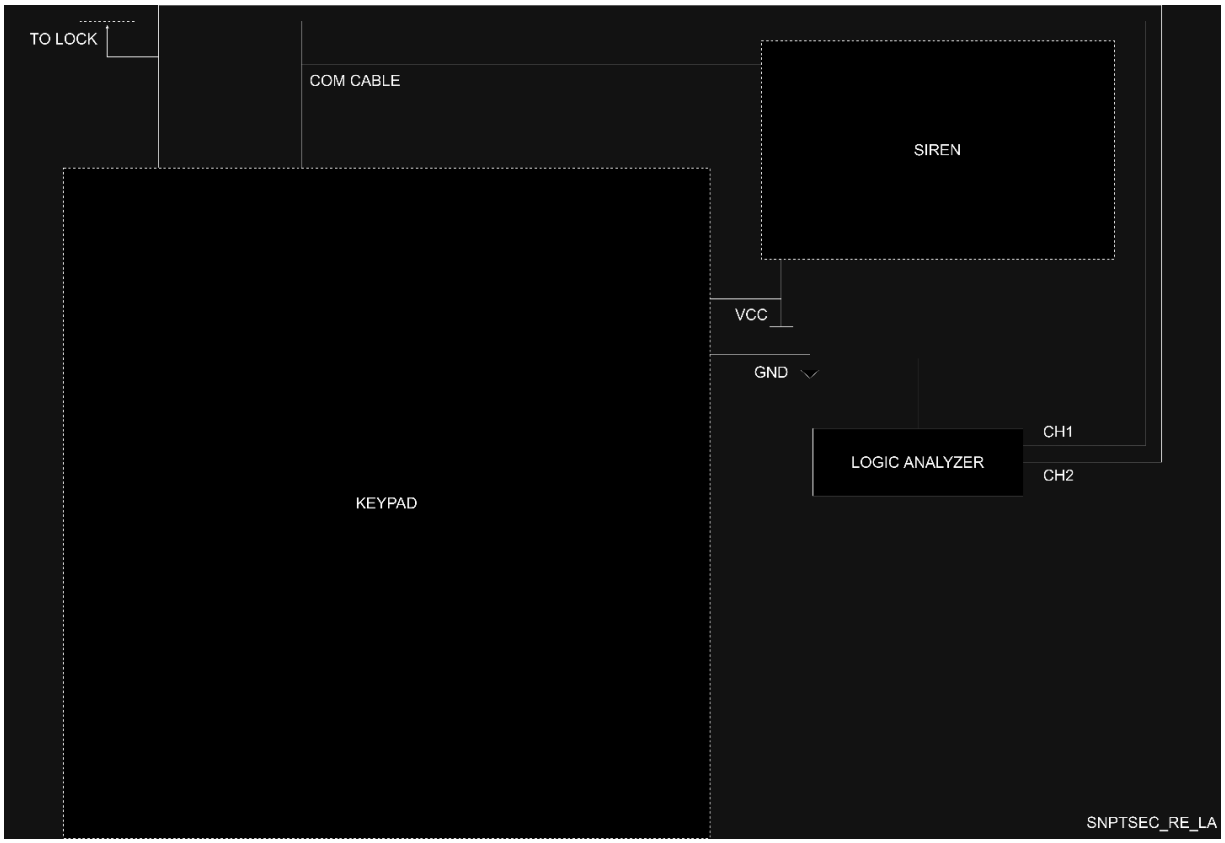

Рисунок 3.9 – Схема підключення логічного аналізатора до системи доступу

Тепер, підключимо логічний аналізатор (рис. 3.10) до ПК, подамо живлення на систему та спробуємо ввести код доступу.

Рисунок 3.10 – Логічний аналізатор, що використовується

Після кількох маніпуляцій з системою отримуємо наступну сигнальну послідовність (рис. 3.11):

Рисунок 3.11 – Зчитана сигнальна послідовність

Застосувавши вбудовану в ПЗ Salee Logic 2 функцію розшифровки сигналів було визначено, що лінія COM, якою проходило спілкування кейпаду та сирени, є звичайною лінією USART (рис. 3.12):

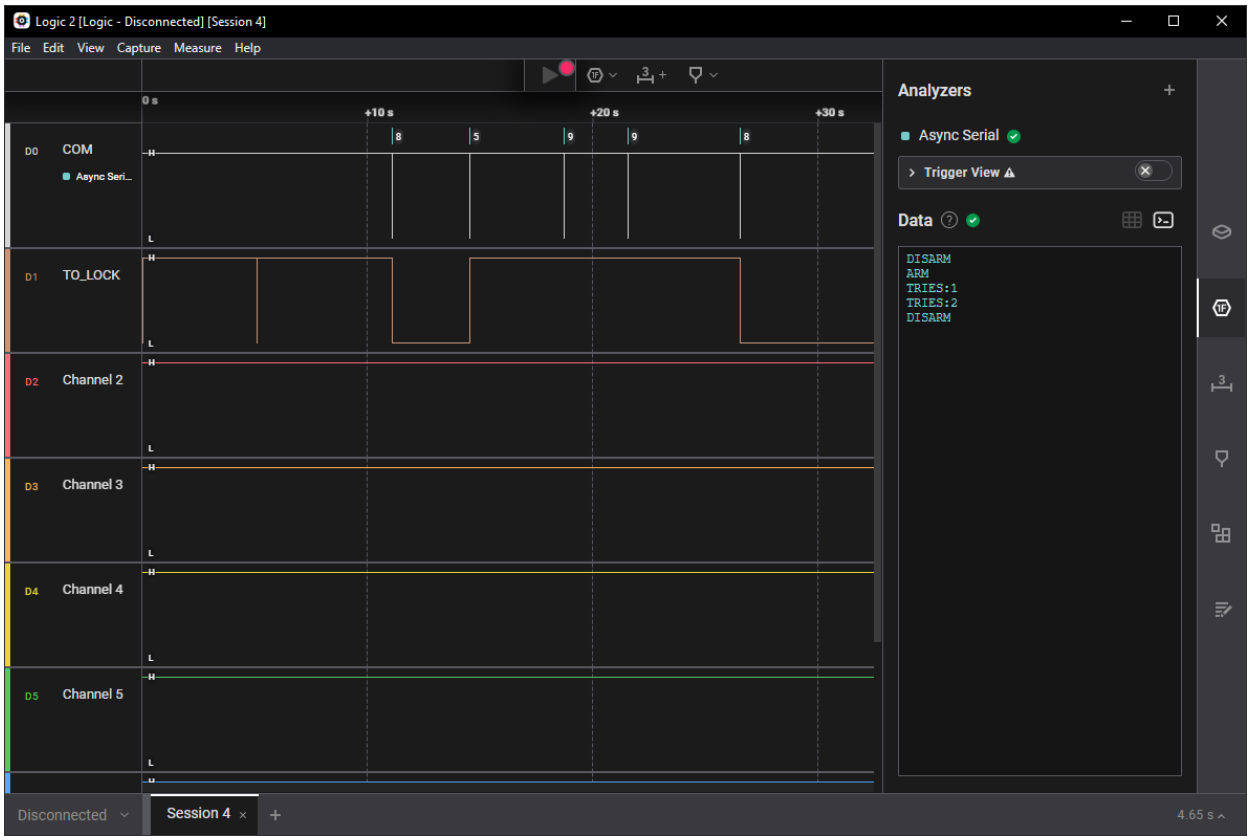

Рисунок 3.12 – Розшифровка перехоплених повідомлень

Також, в правій колонці, ми можемо побачити повністю розшифровані повідомлення, що однозначно означає про відсутність будь-якого захисту не тільки в рамках прошивки пристроїв, а і у рамках протоколу спілкування.

Це означає, що будь-хто, маючи при собі USB-USART перетворювач і термінал на своєму ноутбуці чи мобільному телефоні може запросто вимкнути звукове оповіщення сирени, надіславши їй рядок «DISARM» та заблокувавши таким чином обробку нею будь-яких команд, окрім «ARM». Це може дати зловмиснику вдосталь часу для перебору паролю, без ймовірності своєї компрометації світлозвуковою сигналізацією, якщо було б вирішено не вторгатись в сам пристрій або якщо б у пристрої була присутня додаткова міра захисту в лиці тамперної кнопки (яка б, при відкритті корпусу кейпаду, надіслала команду на сирену про вторгнення та заблокувала б подальшу взаємодію з системою).

Також, співставивши часові проміжки надсилання повідомлень по USART'у та поведінку сигнального кабелю TO_LOCK можна однозначно сказати – сигнальний кабель TO_LOCK подає однобітний логічний сигнал на зовнішній керуючий пристрій, буквально керуючи замком просто фактом наявності/відсутності живлення на лінії (рис. 3.13):

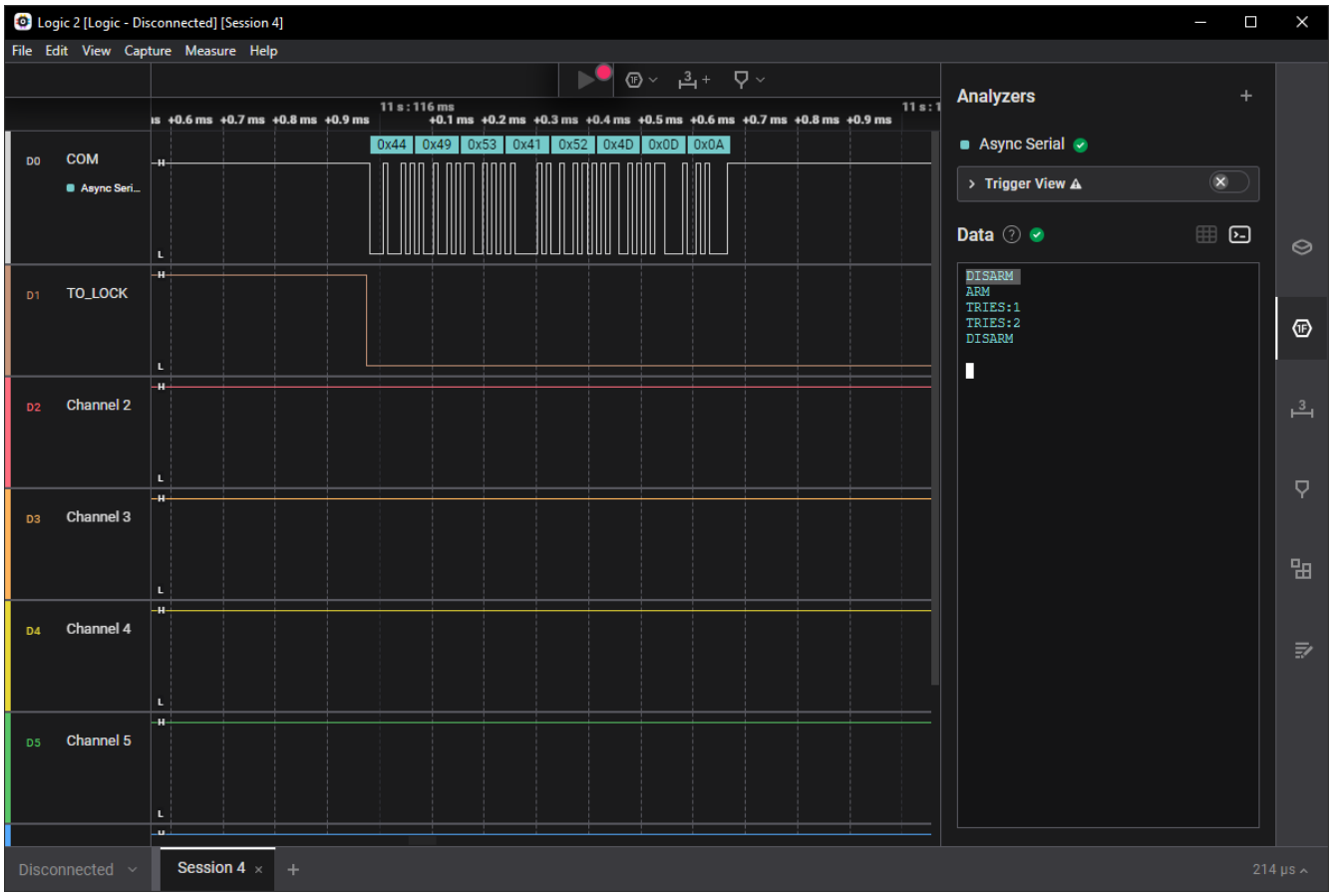

Рисунок 3.13 – Співставлення таймінгів двох сигналів

Такий спосіб реалізації керування є найбільш неприйнятним в подібних системах, адже зловмиснику не треба проводити додаткових маніпуляцій з системою, не потрібно ніяких додаткових знань чи вмінь – зробити PULL_DOWN (або PULL_UP, якби реле відкривалось від позитивного сигналу) по кнопці можна будь-яким звичайним мікроконтролером – AVR, PIC, STM8 – значення немає, потрібен лише сантиметр лінії назовні і півхвилини часу.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ТА МЕТОДІВ ЗАХИСТУ

По результатам незалежного дослідження було встановлено наступні факти:

- Flash-пам’ять системи доступу не була захищеною від читання, а записаний в неї код доступу «лежав» символьною послідовністю;
- Відсутнє шифрування в протоколі спілкування між пристроями;
- Ненадійна реалізація схеми підключення пристрою керування замком.

Цих даних повинно вистачити для початку претензійного процесу проти фірми розробника, що заявляв про «неможливість компрометації» їх системи доступу при передачі продукту замовнику. У зловмисника було кілька варіантів зламу системи, наймовірнішим з яких є зчитування паролю з пам’яті та доступ до об’єкту по ньому, без потреби доступу до прокладених всередині стін кабелів.

На останок постає питання – а чи можна було уникнути подібних наслідків? Чи можна було створити систему, використовуючи стільки ж наявних ресурсів, яка б не була настільки вразливою до зовнішніх атак?

По першому пункту – безумовно так, захист пам’яті пристрою є основою основ при розробці вбудованих систем. Подібна недбалість може дорого коштувати як компанії розробнику (реверс-інжиніринг прошивки давно не є складною задачою для досвідчених спеціалістів компаній-конкурентів), так і для кінцевого покупця (наведений в цій роботі приклад хоч і є вигаданим, але заснований на реальній загрозі, про що буде написано трохи нижче).

По пункту під номером два – розробка протоколу це, безумовно, складне завдання, адже потрібно передбачити не тільки наявні на сьогодні потреби компанії та покупців, але й закласти можливість для масштабування кількості та комплексності майбутніх задач. Але, навіть в такій «одноразовій» системі можна було б уникнути проблем, застосувавши в системі модуль AES (є стандартним периферійним модулем в більшості ARM мікроконтролерів компанії ST) чи просто програмно шифруючи в декілька кроків відомим усім пристроям системи ключем повідомлення. Також, для забезпечення відносної безвідмовності можна було використати протокол передачі RS-485, що є надбудовою над UART’ом і

використовує обидві його лінії для надмірної передачі інформації (друга лінія є інверсною до першої та дозволяє мінімізувати втрати через навколишню електрозашумленість), але це все ж таки потребує додаткових витрат (можливі модулі для цієї задачі зображені на рисунку 4.1).

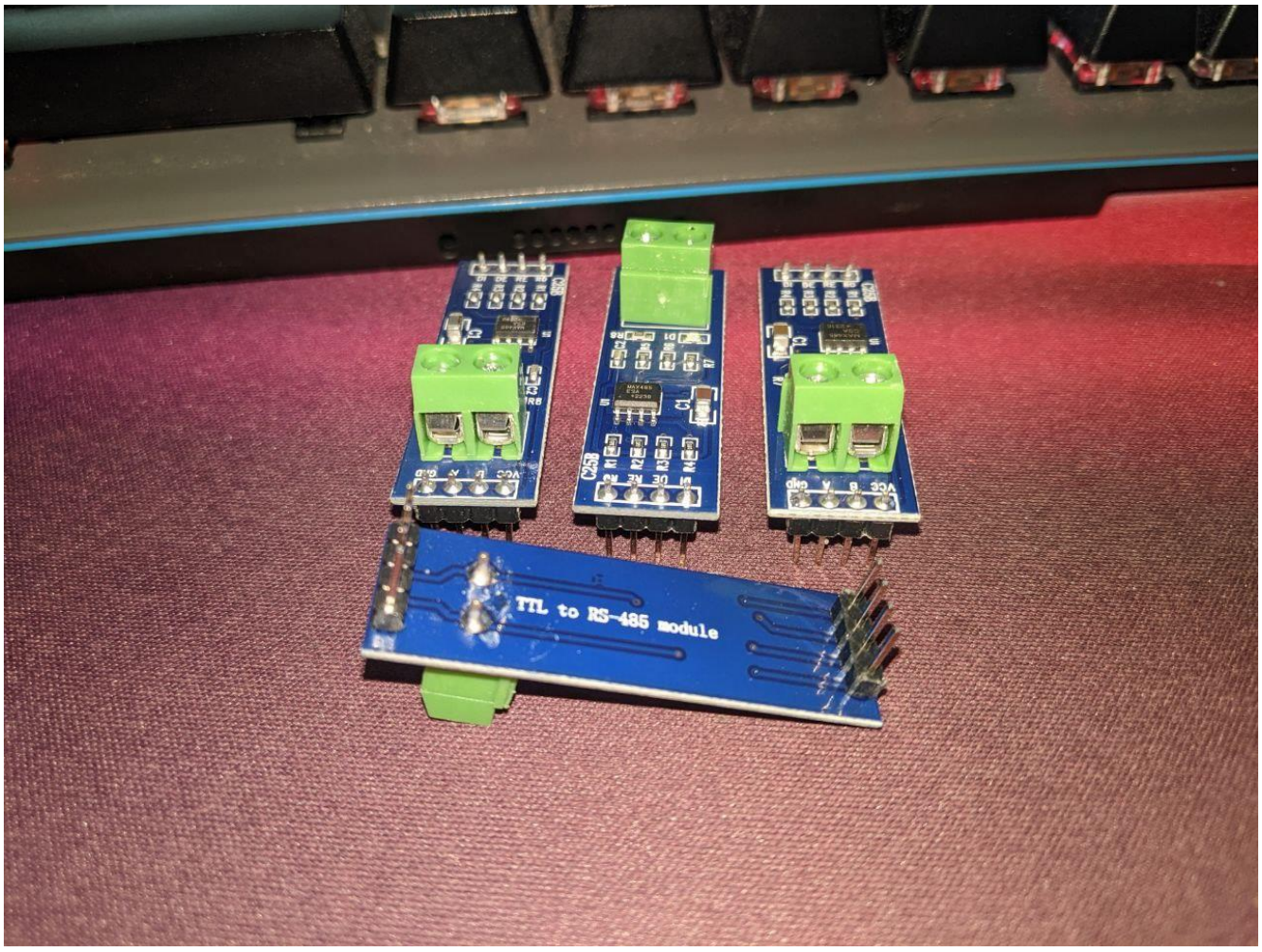

Рисунок 4.1 – Модулі USART – RS-485

І, нарешті, по третьому пункту – уникнути ситуації, описаної в розділі 3, без використання окремого, повноправного в плані спілкування всередині системи, пристрою, що відповідав би тільки за відкриття замку – майже неможливо. Спосіб «підняття піна для відкриття замка» залишається найпопулярнішим.

Нажаль, більшість «доступних» сучасних систем контролю доступу працюють саме таким чином. Візьмемо до прикладу випадкову клавіатуру доступу з китайського маркетплейсу та роздивимось схему підключення до системи доступу (рис. 4.2):

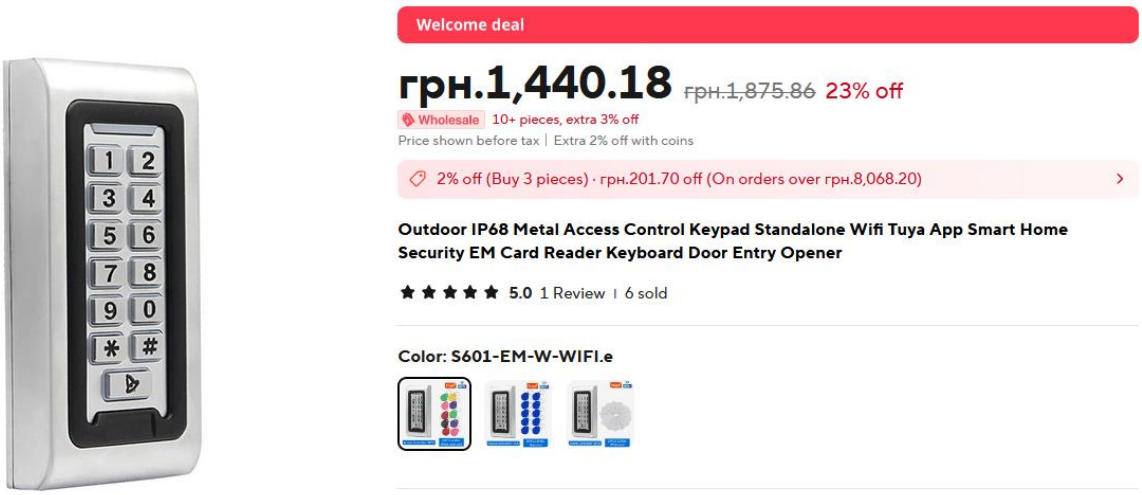

Рисунок 4.2 – Типовий представник бюджетних клавіатур доступу

Рекламні матеріали виглядають наступним чином (рис. 4.3, рис. 4.4) [12]:

Рисунок 4.3 – Діаграма підключення клавіатури доступу в систему

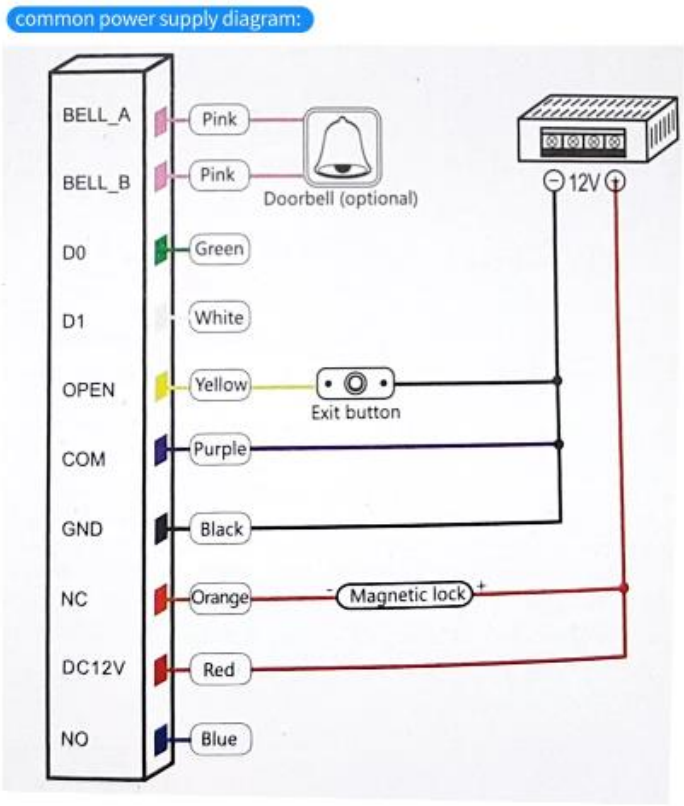

Рисунок 4.4 – Схема з’єднань системи доступу від виробника клавіатур

Тобто, можна зробити висновок, що при недостатній ізоляції лінії COM або при наявності невеличкого лома для зняття кейпаду зі стіни – зловмиснику не складе проблеми відкрити двері всього лиш одним 3.3/5/12V DC сигналом.

Що ж робити потенційному бізнесу чи приватній особі, що воліє захистити свою власність та не бажає одного разу виявити пусту нерухомість без явних слідів зламу?

Один з варіантів – розробити систему самостійно, використовуючи схему підключення пристроїв з рисунку 4.5:

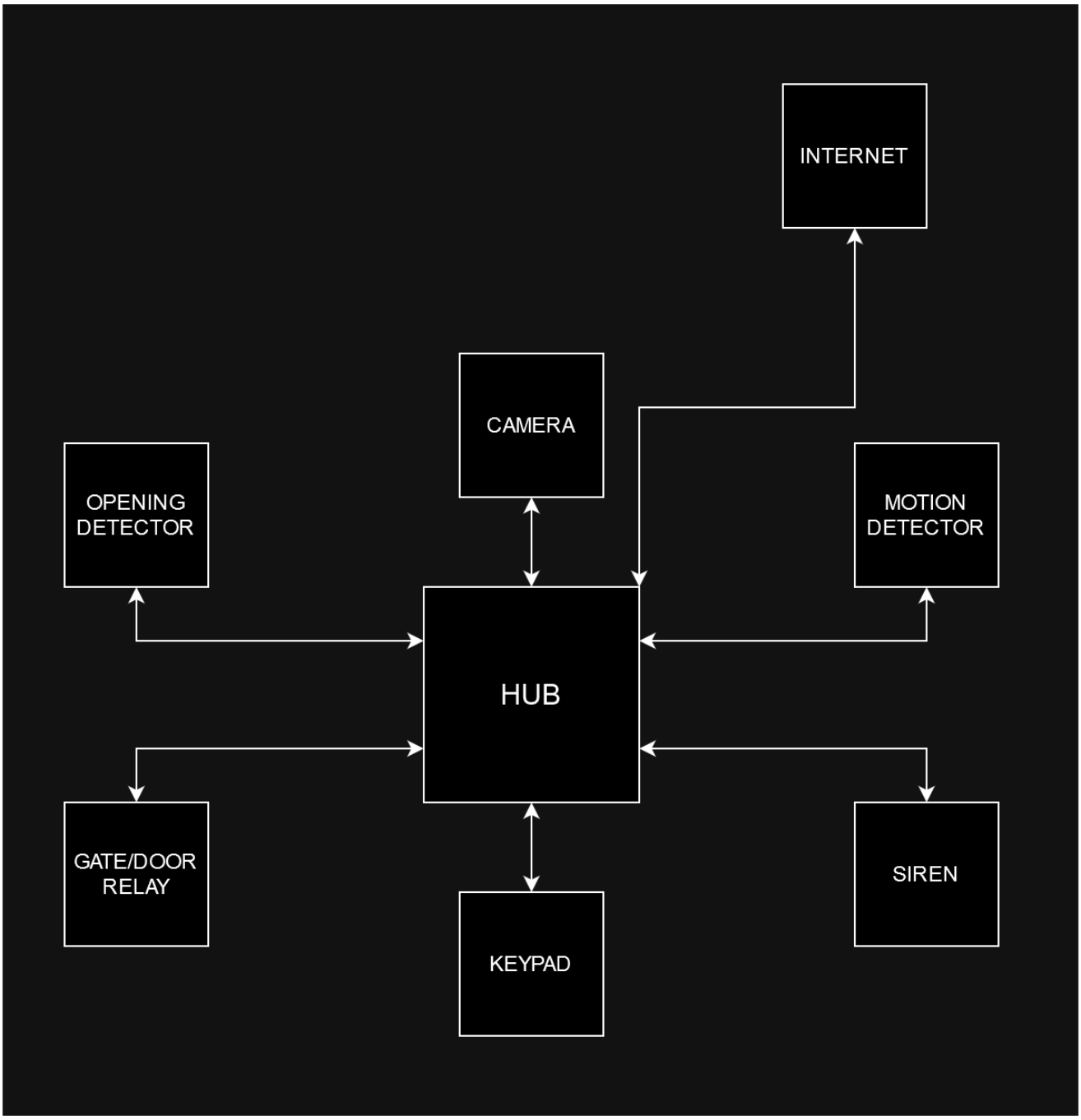

Рисунок 4.5 – Потенційно найкращий варіант реалізації охоронної системи доступу

Система з центральним керуючим пристроєм дозволяє мінімізувати ризики, що пов’язані з компрометацією усієї системи через один з пристроїв. При правильному підході (використання шифрування для спілкування, інтеграція TDMA в процес передачі даних, прийняття рішень виключно на централі, інформування користувача про інциденти безпеки) система стане в рази більш захищена від атак на неї.

Вдалим прикладом реалізації такого підходу є система-на-хабі від української компанії Ajax Systems (на рисунку 4.6 зображений один з пристроїв компанії).

Рисунок 4.6 – Ajax Hub 2, фото з офіційного сайту [15]

Реалізація протоколу спілкування [13] та загальна ідея автономної охоронної системи з можливістю керування зі свого смартфону (чи навіть годинника!)[14], на мій погляд, ставить Ajax на один з найвищих ступенів в плані якості кінцевого продукту та надає приклад як потрібно будувати сучасну охоронну систему.

ВИСНОВОК

Цитуючи вступ документу – «вбудовані системи стали необхідною складовою нашого повсякденного життя, впливаючи на наші звички, зручність та безпеку». Довіряючи свою дані та безпеку (часто, буквально) невеликим малопродуктивним пристроям ми повинні пам’ятати про можливі ризики їх використання та, за можливості, остерігатись їх неконтрольованій інтеграції в наші життя.

В рамках цієї дипломної роботи було проведено наочну демонстрацію важливості безпеки пристроїв вбудованих систем. Розроблена охоронна система доступу показала свою неефективність проти нескладних в реалізації зовнішніх атак. Неспроможність контролю за перебігом подій за межами окремого мікроконтролера та загальне нехтування основами безпечної комунікації між пристроями потенційно можуть призвести до майнових та репутаційних втрат усіх сторін.

На жаль, на момент написання цього висновку, більшість «доступних» охоронних систем доступу все ще містять в собі критичні загрози безпеці, що, як можна побачити з кількості продажів та позитивних відгуків, не висвітлюється на широку публіку.

Розглянуті вразливості є лише «вершиною айсбергу» у світі пристроїв вбудованих систем. Можливі наслідки від недостатньої захищеності embedded систем можна побачити в безлічі новин, починаючи з несанкціонованого підключення до систем відеоспостереження і закінчуючи отриманням віддаленого доступу до систем керування автомобіля[16].

Є надія, що з неминучим збільшенням кількості вбудованих пристроїв у житті кожної людини, також буде збільшуватись обізнаність про потенційні ризики використання таких пристроїв від неперевірених виробників, але точно покаже лише час.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сигнал [Електронний ресурс]. – Режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/133740
2. USB Component: USB Communication [Електронний ресурс]. – Режим доступу: https://www.keil.com/pack/doc/mw/USB/html/_u_s_b__endpoints.html
3. USB Component: USB Protocol [Електронний ресурс]. – Режим доступу: https://www.keil.com/pack/doc/mw/USB/html/_u_s_b__protocol.html
4. Introduction to SPI interface [Електронний ресурс]. – Режим доступу: https://community.nxp.com/t5/MQX-Software-Solutions-Knowledge/Introduction-to-SPI-interface/ta-p/1104630
5. SPI інтерфейс [Електронний ресурс]. – Режим доступу: https://itmaster.biz.ua/directory/standarts/spi.html
6. I2C інтерфейс [Електронний ресурс]. – Режим доступу: https://itmaster.biz.ua/directory/standarts/i2c.html
7. Интерфейс передачи данных - I2C [Електронний ресурс]. – Режим доступу: https://3d-diy.ru/wiki/arduino-moduli/interfeys-peredachi-dannykh-i2c/
8. I2C Primer: What is I2C? [Електронний ресурс]. – Режим доступу: https://www.analog.com/en/resources/technical-articles/i2c-primer-what-is-i2c-part-1.html
9. UART протокол [Електронний ресурс]. – Режим доступу: https://itmaster.biz.ua/directory/standarts/uart.html
10. Интерфейс передачи данных – UART [Електронний ресурс]. – Режим доступу: https://3d-diy.ru/wiki/arduino-moduli/interfeys-peredachi-dannykh-uart/
11. 12-ти кнопкова мембранна клавіатура [Електронний ресурс]. – Режим доступу: https://arduino.ua/prod1133-12-ti-knopochnaya-klaviatyra
12. Outdoor IP68 Metal Access Control Keypad [Електронний ресурс]. – Режим доступу: https://www.aliexpress.com/item/1005005419656956.html

13. Bidirectional security sensor communication system – Google Patents [Електронний ресурс]. – Режим доступу: https://patents.google.com/patent/US10492068B1/en
14. Застосунок Ajax Security System для користувачів [Електронний ресурс]. – Режим доступу: https://ajax.systems/ua/ajax-security-system/
15. Hub 2 Plus – Advanced control panel with 4 communication channels [Електронний ресурс]. – Режим доступу: https://ajax.systems/products/hub2-plus/
16. Hackers Remotely Kill a Jeep on the Highway—With Me in It [Електронний ресурс]. – Режим доступу: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
17. HAL_STM32_I2C_LCD [Електронний ресурс]. – Режим доступу: https://github.com/qazf88/HAL_STM32_I2C_LCD

ДОДАТОК А

Лістинг А.1 – Код файлу main.c:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "lcd.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
// popa
```

```
char passStr[16] = "";
char allowStr[16] = "665";


typedef enum
{
    SIREN_DISARM,
    SIREN_ARM,
    SIREN_ALARM,
    SIREN_WRONG_PASS
}SirenMsg_t;

typedef enum
{
    LCD_DISARM,
    LCD_ARM,
    LCD_WRONG_PASS,
    LCD_NEWCHAR,
    //LCD_CLEAR_PASS,
    LCD_DEFAULT
}DisplayMsg_t;

typedef enum
{
    DOORS_OPEN,
    DOORS_CLOSE
}DoorsCmd_t;

typedef enum
{
    SYS_DISARMED,
    SYS_ARMED
}GlobalState_t;

#define UART_HANDLER huart2

GlobalState_t sysState = SYS_ARMED;
keypad_t keypad;
bool processingPause = false;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C2_Init(void);
/* USER CODE BEGIN PFP */
void ProcessKey(key_pressed key);
void SendSirenMsg(SirenMsg_t msg);
void SendDisplayMgs(DisplayMsg_t msg);
void DoorsCmd(DoorsCmd_t cmd);
void KeyBeep(void);
void DisplayInit(void);

void SetPauseFlag(void);
```

```c
void ResetPauseFlag(void);
bool IsProcessingPaused(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C2_Init();
  /* USER CODE BEGIN 2 */
  keypad.key = KEY_NONE;
  keypad.keyPending = false;
  DisplayInit();
  DoorsCmd(DOORS_CLOSE);
  // LED RED
  HAL_GPIO_WritePin(LD5_GPIO_Port, LD5_Pin, GPIO_PIN_SET);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
```

```
    /* USER CODE BEGIN 3 */
      HAL_Delay(100);
      if(IsProcessingPaused())
      {
         static uint8_t pause;
         pause++;
         if(pause >= 30) // 3 sec pause
         {
            ResetPauseFlag();
            keypad.keyPending = false;
            pause = 0;
            SendDisplayMgs(LCD_DEFAULT);
         }
      }
      else
      {
         if(keypad.keyPending)
         {
            ProcessKey(keypad.key);
            keypad.keyPending = false;
         }
      }
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.ClockSpeed = 100000;
  hi2c2.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
```

```
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOE, KEY_ROW_0_Pin|KEY_ROW_1_Pin|KEY_ROW_2_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(Relay_GPIO_Port, Relay_Pin, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pins : KEY_ROW_0_Pin KEY_ROW_1_Pin KEY_ROW_2_Pin */
  GPIO_InitStruct.Pin = KEY_ROW_0_Pin|KEY_ROW_1_Pin|KEY_ROW_2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pin : BOOT1_Pin */
  GPIO_InitStruct.Pin = BOOT1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin */
  GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pin : Relay_Pin */
  GPIO_InitStruct.Pin = Relay_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(Relay_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PC7 PC9 */
  GPIO_InitStruct.Pin = GPIO_PIN_7|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : KEY_0_Pin KEY_1_Pin KEY_2_Pin KEY_3_Pin */
  GPIO_InitStruct.Pin = KEY_0_Pin|KEY_1_Pin|KEY_2_Pin|KEY_3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

  HAL_NVIC_SetPriority(EXTI1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI1_IRQn);

  HAL_NVIC_SetPriority(EXTI2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI2_IRQn);

  HAL_NVIC_SetPriority(EXTI3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI3_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

```
/* USER CODE BEGIN 4 */

void SetPauseFlag(void)
{
    processingPause = true;
}

void ResetPauseFlag(void)
{
    processingPause = false;
}

bool IsProcessingPaused(void)
{
    return processingPause;
}

void ProcessKey(key_pressed key)
{
#define MAX_PASS_LENGTH 10
    static uint8_t passLength = 0;
    KeyBeep();
    switch(sysState)
    {
    case SYS_DISARMED:
       if(key == KEY_RESH)
       {
          sysState = SYS_ARMED;
          HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_RESET);
          HAL_GPIO_WritePin(LD5_GPIO_Port, LD5_Pin, GPIO_PIN_SET);
          DoorsCmd(DOORS_CLOSE);
          SendSirenMsg(SIREN_ARM);
          SendDisplayMgs(LCD_ARM);
          SetPauseFlag();
       }
       break;
    case SYS_ARMED:
       if((key != KEY_ZIR) && (key != KEY_RESH))
       {
          if(passLength < MAX_PASS_LENGTH)
          {
             passLength++;
             char newSym = key;
             strncat(passStr, &newSym, 1);
             SendDisplayMgs(LCD_NEWCHAR);
          }
          else
          {
             do{}while(0);
          }
       }
       else
       {
          if(key == KEY_RESH)
          {
             if(!strcmp(passStr, allowStr))
```

```
            {
               sysState = SYS_DISARMED;
               HAL_GPIO_WritePin(LD5_GPIO_Port, LD5_Pin, GPIO_PIN_RESET);
               HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);
               DoorsCmd(DOORS_OPEN);
               SendDisplayMgs(LCD_DISARM);
               SendSirenMsg(SIREN_DISARM);
               SetPauseFlag();
            }
            else if(strlen(passStr))
            {
               SendDisplayMgs(LCD_WRONG_PASS);
               SendSirenMsg(SIREN_WRONG_PASS);
               SetPauseFlag();
            }

         }
         else if(key == KEY_ZIR)
         {
            if(strlen(passStr))
               SendDisplayMgs(LCD_DEFAULT);
         }
         memset(passStr,0,strlen(passStr));
         passLength = 0;
      }
      break;
   }
}

void SendSirenMsg(SirenMsg_t msg)
{
   static uint8_t tries = 0;
   char alarmStr[20] = "";

   switch(msg)
   {
      case SIREN_DISARM:
         tries = 0;
         sprintf(alarmStr, "DISARM\r\n");
         break;
      case SIREN_ARM:
         sprintf(alarmStr, "ARM\r\n");
         break;
      case SIREN_WRONG_PASS:
         tries++;
         sprintf(alarmStr, "TRIES:%d\r\n", tries);
         break;
      case SIREN_ALARM:
         sprintf(alarmStr, "ALARM\r\n");
         break;
   }
   HAL_UART_Transmit(&UART_HANDLER, (uint8_t *)alarmStr, strlen(alarmStr), 500);
}

void KeyBeep(void)
{
```

```
    // GPIO_HIGH
    HAL_Delay(150);
    // GPIO LOW
}

void DisplayInit(void)
{
    // LCD1602 init
    // Cursor to first line
    // send "SNPT ALARM V1.1"
    // Cursor to second line
    LCD_init(&hi2c2, 0x27);
    LCD_clear();
    HAL_Delay(50);
    SendDisplayMgs(LCD_DEFAULT);

}

void SendDisplayMgs(DisplayMsg_t msg)
{
    char dispStr[20] = "";
    static uint8_t prevPos = 1;
    switch(msg)
    {
       case LCD_DISARM:
          LCD_clear();
          memset(dispStr, 0, strlen(dispStr));
          sprintf(dispStr, "Access Granted");
          LCD_write_str(dispStr);
          prevPos = 1;
          break;
       case LCD_ARM:
          LCD_clear();
          memset(dispStr, 0, strlen(dispStr));
          sprintf(dispStr, "Armed");
          LCD_write_str(dispStr);
          break;
       case LCD_WRONG_PASS:
          LCD_clear();
          memset(dispStr, 0, strlen(dispStr));
          sprintf(dispStr, "Pass Incorrect");
          LCD_write_str(dispStr);
          prevPos = 1;
          break;
       case LCD_NEWCHAR:
          LCD_set_pos(prevPos, 1);
          memset(dispStr, 0, strlen(dispStr));
          sprintf(dispStr, "*");
          LCD_write_str(dispStr);
          prevPos++;
          break;
       case LCD_DEFAULT:
          if(sysState == SYS_ARMED)
          {
             prevPos = 1;
             LCD_clear();
```

```
            memset(dispStr, 0, strlen(dispStr));
            LCD_set_pos(0, 0);
            sprintf(dispStr, "Enter pass:");
            LCD_write_str(dispStr);

            memset(dispStr, 0, strlen(dispStr));
            LCD_set_pos(0, 1);
            sprintf(dispStr, ">");
            LCD_write_str(dispStr);
         }
         else if(sysState == SYS_DISARMED)
         {
            memset(dispStr, 0, strlen(dispStr));
            LCD_set_pos(0, 0);
            sprintf(dispStr, "SNPT SECURITY ");
            LCD_write_str(dispStr);

            memset(dispStr, 0, strlen(dispStr));
            LCD_set_pos(0, 1);
            sprintf(dispStr, "DISARMED");
            LCD_write_str(dispStr);
         }
         break;
    }
    // LCD push
}

void DoorsCmd(DoorsCmd_t cmd)
{
    switch(cmd)
    {
       case DOORS_OPEN:
          HAL_GPIO_WritePin(Relay_GPIO_Port, Relay_Pin, GPIO_PIN_RESET);
          break;
       case DOORS_CLOSE:
          HAL_GPIO_WritePin(Relay_GPIO_Port, Relay_Pin, GPIO_PIN_SET);
          break;
    }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Лістинг А.2 – Код файлу stm32f4xx_it.c:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    stm32f4xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */
```

```
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/

/* USER CODE BEGIN EV */
extern keypad_t keypad;
bool sec_it = false;

/* USER CODE END EV */

/******************************************************************************/
/*           Cortex-M4 Processor Interruption and Exception Handlers          */
/******************************************************************************/
/**
  * @brief This function handles Non maskable interrupt.
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
   while (1)
  {
  }
  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
  * @brief This function handles Hard fault interrupt.
  */
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
```

```
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
      HAL_GPIO_TogglePin(LD5_GPIO_Port, LD5_Pin);
      HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
}

/**
  * @brief This function handles Memory management fault.
  */
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
    /* USER CODE END W1_MemoryManagement_IRQn 0 */
  }
}

/**
  * @brief This function handles Pre-fetch fault, memory access fault.
  */
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
}

/**
  * @brief This function handles Undefined instruction or illegal state.
  */
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
  }
}

/**
  * @brief This function handles System service call via SWI instruction.
  */
```

```
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
  * @brief This function handles Debug monitor.
  */
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
  * @brief This function handles Pendable request for system service.
  */
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32F4xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                    */
/******************************************************************************/
```

```c
/**
  * @brief This function handles EXTI line0 interrupt.
  */
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(KEY_0_Pin);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
  if(sec_it)
  {
      sec_it = false;
      return;
  }

  if(keypad.keyPending)
  {
      return;
  }

  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
     keypad.key = KEY_1;
     keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
      keypad.key = KEY_2;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_0_GPIO_Port, KEY_0_Pin))
  {
      keypad.key = KEY_3;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
  /* USER CODE END EXTI0_IRQn 1 */
}
```

```c
/**
  * @brief This function handles EXTI line1 interrupt.
  */
void EXTI1_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI1_IRQn 0 */

  /* USER CODE END EXTI1_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(KEY_1_Pin);
  /* USER CODE BEGIN EXTI1_IRQn 1 */
  if(sec_it)
  {
      sec_it = false;
      return;
  }

  if(keypad.keyPending)
  {
      return;
  }

  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_1_GPIO_Port, KEY_1_Pin))
  {
      keypad.key = KEY_4;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_1_GPIO_Port, KEY_1_Pin))
  {
      keypad.key = KEY_5;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_1_GPIO_Port, KEY_1_Pin))
  {
      keypad.key = KEY_6;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
  /* USER CODE END EXTI1_IRQn 1 */
}
```

```
/**
  * @brief This function handles EXTI line2 interrupt.
  */
void EXTI2_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI2_IRQn 0 */

  /* USER CODE END EXTI2_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(KEY_2_Pin);
  /* USER CODE BEGIN EXTI2_IRQn 1 */
  /* USER CODE BEGIN EXTI1_IRQn 1 */
  if(sec_it)
  {
      sec_it = false;
      return;
  }

  if(keypad.keyPending)
  {
      return;
  }

  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_2_GPIO_Port, KEY_2_Pin))
  {
      keypad.key = KEY_7;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_2_GPIO_Port, KEY_2_Pin))
  {
      keypad.key = KEY_8;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_2_GPIO_Port, KEY_2_Pin))
  {
      keypad.key = KEY_9;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
  /* USER CODE END EXTI2_IRQn 1 */
```

```
}

/**
  * @brief This function handles EXTI line3 interrupt.
  */
void EXTI3_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI3_IRQn 0 */

  /* USER CODE END EXTI3_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(KEY_3_Pin);
  /* USER CODE BEGIN EXTI3_IRQn 1 */
  if(sec_it)
  {
      sec_it = false;
      return;
  }

  if(keypad.keyPending)
  {
      return;
  }

  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_3_GPIO_Port, KEY_3_Pin))
  {
      keypad.key = KEY_ZIR;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_0_GPIO_Port, KEY_ROW_0_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_3_GPIO_Port, KEY_3_Pin))
  {
      keypad.key = KEY_0;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_1_GPIO_Port, KEY_ROW_1_Pin, GPIO_PIN_SET);

  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_RESET);
  if(!HAL_GPIO_ReadPin(KEY_3_GPIO_Port, KEY_3_Pin))
  {
      keypad.key = KEY_RESH;
      keypad.keyPending = true;
     sec_it = true;
     HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
     return;
  }
  HAL_GPIO_WritePin(KEY_ROW_2_GPIO_Port, KEY_ROW_2_Pin, GPIO_PIN_SET);
  /* USER CODE END EXTI3_IRQn 1 */
```

```
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

Лістинг А.3 – Код файлу main.h:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
```

```
/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */
typedef enum
{
    KEY_NONE = 0x00,
    KEY_1    = 0x31,
    KEY_2    = 0x32,
    KEY_3    = 0x33,
    KEY_4    = 0x34,
    KEY_5    = 0x35,
    KEY_6    = 0x36,
    KEY_7    = 0x37,
    KEY_8    = 0x38,
    KEY_9    = 0x39,
    KEY_ZIR  = 0x2A,
    KEY_0    = 0x30,
    KEY_RESH = 0x23
}key_pressed;

typedef struct
{
    key_pressed key;
    bool keyPending;
}keypad_t;
/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define KEY_ROW_0_Pin GPIO_PIN_4
#define KEY_ROW_0_GPIO_Port GPIOE
#define KEY_ROW_1_Pin GPIO_PIN_5
#define KEY_ROW_1_GPIO_Port GPIOE
#define KEY_ROW_2_Pin GPIO_PIN_6
#define KEY_ROW_2_GPIO_Port GPIOE
#define PC14_OSC32_IN_Pin GPIO_PIN_14
#define PC14_OSC32_IN_GPIO_Port GPIOC
#define PC15_OSC32_OUT_Pin GPIO_PIN_15
#define PC15_OSC32_OUT_GPIO_Port GPIOC
#define PH0_OSC_IN_Pin GPIO_PIN_0
#define PH0_OSC_IN_GPIO_Port GPIOH
#define PH1_OSC_OUT_Pin GPIO_PIN_1
#define PH1_OSC_OUT_GPIO_Port GPIOH
#define BOOT1_Pin GPIO_PIN_2
#define BOOT1_GPIO_Port GPIOB
#define LD4_Pin GPIO_PIN_12
#define LD4_GPIO_Port GPIOD
#define LD3_Pin GPIO_PIN_13
#define LD3_GPIO_Port GPIOD
```

```
#define LD5_Pin GPIO_PIN_14
#define LD5_GPIO_Port GPIOD
#define LD6_Pin GPIO_PIN_15
#define LD6_GPIO_Port GPIOD
#define Relay_Pin GPIO_PIN_6
#define Relay_GPIO_Port GPIOC
#define SWDIO_Pin GPIO_PIN_13
#define SWDIO_GPIO_Port GPIOA
#define SWCLK_Pin GPIO_PIN_14
#define SWCLK_GPIO_Port GPIOA
#define KEY_0_Pin GPIO_PIN_0
#define KEY_0_GPIO_Port GPIOD
#define KEY_0_EXTI_IRQn EXTI0_IRQn
#define KEY_1_Pin GPIO_PIN_1
#define KEY_1_GPIO_Port GPIOD
#define KEY_1_EXTI_IRQn EXTI1_IRQn
#define KEY_2_Pin GPIO_PIN_2
#define KEY_2_GPIO_Port GPIOD
#define KEY_2_EXTI_IRQn EXTI2_IRQn
#define KEY_3_Pin GPIO_PIN_3
#define KEY_3_GPIO_Port GPIOD
#define KEY_3_EXTI_IRQn EXTI3_IRQn
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

Повний код використаної бібліотеки lcd (файли lcd.c та lcd.h) можна знайти за посиланням [17].