КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від ____ травня 2024 року, протокол №_____.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

ДИПЛОМНА РОБОТА МАГІСТРА

на тему:

«РОЗПІЗНАВАННЯ ОБРАЗІВ НА МІКРОКОНТРОЛЕРІ»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОНП «Інформаційна безпека телекомунікаційних систем і мереж»
Баханов Іван Олексійович _____

Науковий керівник:
канд. фіз.-мат. наук, асистент
Фесенко Сергій Олександрович _____

Рецензент:
доктор технічних наук, професор
завідувач кафедри технології та конструкційних матеріалів і матеріалознавства
факультету конструювання та дизайну
Національного університету біоресурсів і природокористування України

Лопатько Костянтин Георгійович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Іван БАХАНОВ

Київ − 2024

РЕФЕРАТ

Дипломна робота: 44 с., 15 рис., 1 дод. (14с.), 15 джерел.

ШВИДКЕ ПЕРЕТВОРЕННЯ ФУР'Є, РОЗПІЗНАВАННЯ МОВЛЕННЯ, ЯДРО CORTEX-M3, ОПЕРАЦІЙНИЙ ПІДСИЛЮВАЧ.

Об'єкт розроблення – алгоритм розпізнавання мовлення на основі мікроконтролера STM32F103C8T6.

Мета роботи – перевірка принципової можливості побудови системи розпізнавання мовлення на основі мікроконтролера STM32F103C8T6.

Розроблено систему розпізнавання мовлення на основі мікроконтролера STM32F103C8T6. Запропонований алгоритм побудований на основі дослідження спектрального складу фрази, що розпізнається. Побудована у даній роботі система розпізнає лише дві фрази, які керують станом світлодіода, однак алгоритм дозволяє необмежено розширити кількість таких фраз і можливості керування.

Робота присвячена експериментальній перевірці можливості побудови систем керування на основі голосових команд із використанням порівняно дешевих 32 розрядних мікроконтролерів, які стали доступними останнім часом.

В результаті проведеної роботи можна впевнено стверджувати про можливість успішної побудови систем голосового керування на мікроконтролерах ARM (CORTEX-M3) без залучення обміну даними із зовнішнім сервером. Однак, побудована система є досить примітивною, тому вона може успішно розпізнавати лише окремі фрази, а не виділяти їх з потоку мовлення. Також, імовірно, будуть проблеми з розпізнаванням за наявності значного шумового сигналу, але з цієї точки зору дослідження не проводились.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

АОО — Алгоритм Опорних Обертонів

ІМС — Інтегральна МікроСхема

ОП — Операційний Підсилювач

АЦП — Аналогово-Цифровий перетворювач

DMA – прямий доступ до пам’яті

GANs — генеративно-змагальні мережі

BERT — Bidirectional Encoder Representations from Transformers

GPT — Generative Pre-trained Transformer

RNN – рекурентні нейронні мережі

LSTM – Long Short-Term Memory

ResNet – мережі з резидуальними зв'язками

ViTs – Vision Transformers

ВСТУП

Розпізнавання образів є однією з найважливіших і найбільш перспективних технологій сучасності, що має широке застосування в різних сферах, таких як промислова автоматизація, робототехніка, системи безпеки та інтелектуальні пристрої. З розвитком мікроелектроніки з’явилася необхідність та мініатюризацією обчислювальних компонентів стало можливим впровадження алгоритмів розпізнавання образів на мікроконтролерах. Це створює нові можливості для розробки автономних систем, які можуть виконувати завдання в режимі реального часу без необхідності підключення до потужних обчислювальних центрів.

Мікроконтролери, завдяки своїй енергоефективності, компактності та достатній обчислювальній потужності, стали основними елементами багатьох сучасних електронних пристроїв. Впровадження технологій розпізнавання образів на мікроконтролери дозволяє створювати більш інтелектуальні та автономні системи, такі як смарт-камери, системи контролю доступу, автоматичні транспортні засоби та медичні пристрої.

У цій роботі розглядаються основні методи та алгоритми розпізнавання образів, які можуть бути реалізовані на сучасних мікроконтролерах. Особлива увага приділяється вибору оптимальних алгоритмів, які забезпечують баланс між точністю розпізнавання та ефективним використанням обчислювальних ресурсів. Також аналізуються питання енергоспоживання, швидкості обробки даних та можливостей інтеграції з іншими компонентами системи.

Дослідження в цій галузі має значний практичний інтерес, оскільки воно дозволяє розширити функціональність існуючих пристроїв та створити нові, більш досконалі технологічні рішення, що підвищують ефективність і безпеку в різних сферах діяльності.

1 АЛГОРИТМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

Існують різні підходи вирішення задач розпізнавання образів, зокрема і більш вузької задачі розпізнавання мовлення. Останнім часом стали дуже потужними системи побудовані на основі нейронних мереж. Однак, окрім нейронних мереж для вирішення подібних задач можна використовувати і підходи засновані на аналізі спектру. Певні аспекти згаданих підходів описані нижче.

1.1 Розпізнавання мовлення на основі аналізу спектру.

У роботі [1] побудована система розпізнавання музичних нот на основі спектрального аналізу аудіо-даних. Тут розглядаються різні варіанти розпізнавання, як то розрахунок функції кореляції невідомих даних з опорним сигналом, так і перетворення Фур'є, що, на думку авторів, є кращим варіантом.

Розпізнавання образів – це віднесення вихідних даних до певного класу за допомогою виділення суттєвих ознак, що характеризують ці дані, з загальної маси несуттєвих даних .

Гармонічний ряд (або обертоновий ряд) – звукоряд, що виникає в результаті спонтанного поділу струни на рівні частини під час коливання . Спектр звуку – це частотне представлення складу звуку. Спектр звуку зазвичай представляють на координатній площині, де по осі абсцис відкладена частота, а по осі ординат – амплітуда або інтенсивність кожної гармонічної складової звуку.

Кількість частин, на які спонтанно ділиться струна, в загальному випадку не обмежена. Джерело звуку, що коливається повністю, виробляє основну частоту, найчутніший звук, який здається єдиним. Другі частини (половини) виробляють звук з частотою вдвічі більшою, ніж основна; треті частини – втричі, четверті – вчетверо більшою, ніж основна, і т.д. В результаті таких складних одночасних коливань і виникає звук складного складу [2]. На рисунку 1.1 наведений приклад

спектра звуку, отриманого при виконанні на трубі звуку «до» другої октави (525.478 Гц) та показані опорні обертони.

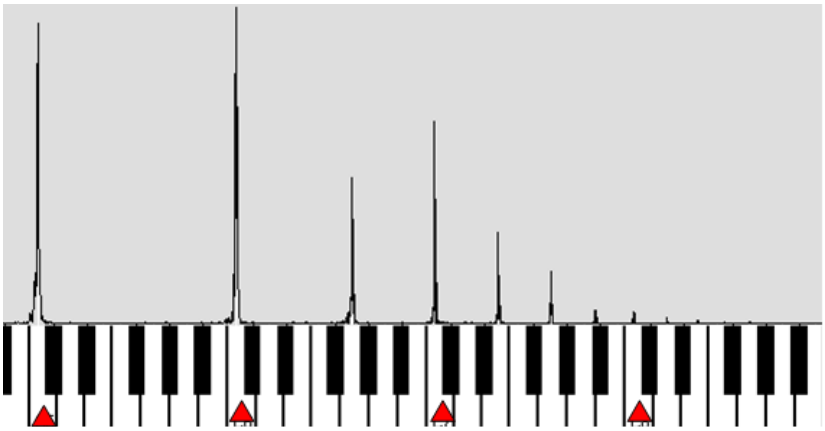

Рисунок 1.1 Спектр звуку, отриманого при виконанні на трубі звуку «до» другої октави [1].

Необхідно зазначити, що під «струною, яка коливається» автори розуміють джерело звуку музичного інструмента. Для скрипки, рояля – це струна, для голосу – голосові зв’язки тощо.

Існують різні підходи до реалізації спектрального аналізу. Для розпізнавання неможливо використовувати всі обертони, оскільки їх безкінечна кількість. Джерело звуку може спонтанно ділитися на безліч частин, що призводить до появи багатьох гармонік у його спектрі. Тому на практиці використовують не всі обертони, а лише їх частину. Рисунок 1.1 свідчить про те, що кожен обертон має свою певну амплітуду (характеристика амплітуд гармонічних складових формує тембр), а амплітуда верхніх обертонів розглянутого спектру дуже мала порівняно з амплітудою основного тону (найнижча гармоніка на рисунку 1.1 – ліворуч).

Для розпізнавання можуть використовуватися всі наступні обертони поспіль або певна вибірка. Розпізнавання за алгоритмом опорних обертонів (далі – АОО) передбачає підсумовування значень амплітуд.

Оскільки гармоніки (обертони) 1, 2, 4, 8 і 16 мають однакові назви (на рисунку 1.1 це нота «до» першої, другої, третьої і четвертої октав; відзначені червоними трикутниками), то їх можна прийняти за основу. Таку операцію необхідно провести над усіма звуками, які потрібно шукати у спектрі. Загалом автори виділяють наступні кроки, які необхідні для розпізнавання нот:

1) встановити порогове значення амплітуди;

2) знайти в спектрі гармоніки, значення амплітуди яких більше, ніж встановлене порогове значення в п.1. Ноти, що відповідають знайденим гармонікам, будуть перевірятися у спектрі;

3) виконати обчислення для кожної ноти з п.2: обчислити суму рівнів амплітуд для всіх однойменних тонів (опорних обертонів);

4) знайти суму з максимальним значенням. Нота, для якої ця сума була обчислена, стане основною нотою даного спектра.

Приклади, що демонструють роботу алгоритму під час гри духового оркестру зображені на рисунках 1.2 та 1.3 (оркестр грає "до мажорне" тризвуччя з нотою "соль" у мелодійному голосі). Вертикальними лініями позначено положення "опорних" обертонів.

1)Порогове значення амплітуди дорівнює 7.

2) Гармоніки, амплітуда яких вище, ніж 7: "ми другої октави" та "соль другої октави".

3) Сума для "ми другої октави" = 8 + 2 + 0 + 0 = 10; Сума для "соль другої октави" = 10 + 4 + 1 + 0 = 16;

4) Максимальне значення дорівнює 16, яке було виявлено для ноти "соль другої октави". Другий алгоритм діє подібно до першого, за винятком того, що для розпізнавання використовується 16 поспіль наступних обертонів.

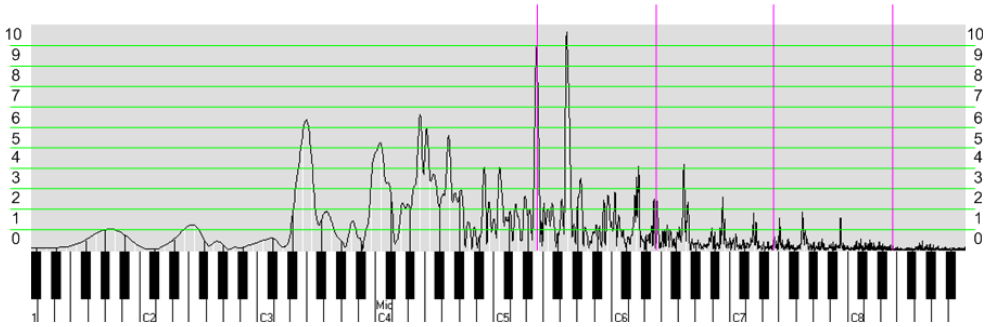

Рисунок 1.2 Аналіз спектру за алгоритмом опорних обертонів від ноти «мі другої октави» [1].

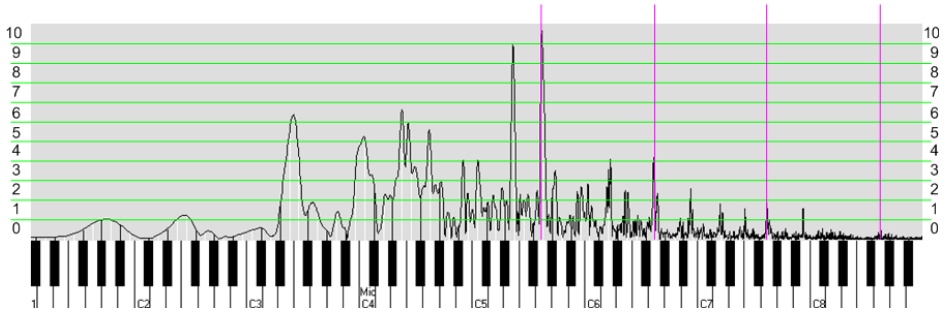

Рисунок 1.3 Аналіз спектру за алгоритмом опорних обертонів від ноти «соль малої октави» [1].

При аналізі нестаціонарних сигналів застосовується перетворення Фур'є з певним часовим вікном. Повний часовий сигнал розділяється на підінтервали - часові вікна, і перетворення проводиться для кожного вікна окремо. Відповідно до тривалості вікна і ноти, яка потрапила в часовий інтервал, можна розрізняти: коротке, велике і еквівалентне вікна Фур'є. Вибір короткого вікна Фур'є призводить до того, що ноти, довші, ніж обраний часовий інтервал, будуть розбиті на більш дрібні, що призведе до захоплення зайвих нот..

Вибір надто великого вікна Фур'є також призводить до помилок розпізнавання, оскільки спектр стає надто розмитим, що проілюстровано на рисунку 1.4.

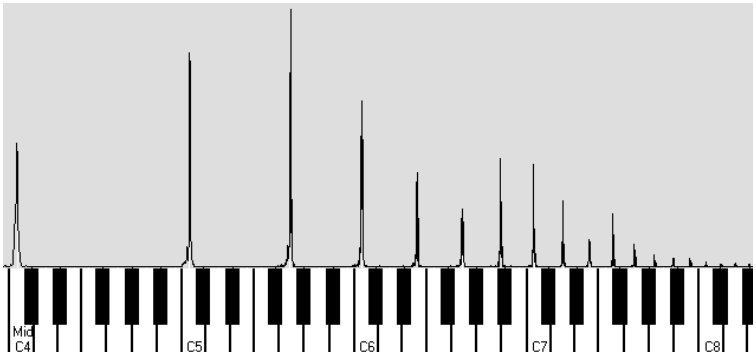

Рисунок 1.4 – Ілюстрація втрати гармонік ноти "мі", що викликано використанням великого вікна Фур'є [1].

З вищезазначеного випливає, що вікно Фур'є повинно мати змінний розмір при розпізнаванні і збігатись з початком і кінцем звучання кожної ноти, яку розпізнаємо. Отже, застосування методів розпізнавання нот, заснованих на віконному перетворенні Фур'є, має проблеми, пов'язані з особливостями аналізу нестаціонарних сигналів. Для вирішення таких проблем автори пропонують наступні підходи:

1) застосування вікон Фур'є змінної довжини. Для визначення розміру вікна, можливо, знадобиться попередній аналіз сигналу.

2) Використання обмеження за частотою або тембром при використанні вікон.

3) Застосування дискретних вейвлет-перетворень, які дозволяють перевірити наявність гармоніки на заданому відрізку часу.

4) Використання "словників музичних фраз". Аналогічно до програм розпізнавання мови та тексту, які ґрунтуються на своєму внутрішньому словнику, можна діяти із музичними композиціями.

5) Застосування нейронної мережі для знаходження меж переходів між нотами або акордами, а також як механізм розпізнавання.

Основною перевагою викладеного підходу є те, що результат пошуку за нотами не залежить від тембру, темпу та тональності записаного аудіофрагменту

в файлі та ключового зразка, що досягається застосуванням розроблених алгоритмів.

1.2 Загальні принципи побудови нейронних мереж та класифікація їх основних типів

Як вже зазначалось, для розпізнавання образів можна успішно використовувати нейронні мережі, які складаються з основних компонентів, що включають нейрони, синапси та суматори. Нейрони, як основні елементи обробки інформації, виконують різноманітні обчислення та активації в залежності від вхідних сигналів. Кожен нейрон приймає вхідні сигнали через свої дендрити, які потім обробляються у внутрішній частині нейрону — сомі [3]. Сома нейрону обчислює ваговану суму вхідних сигналів та передає отриману інформацію до аксона. Аксон нейрону відповідає за передачу сигналів до інших нейронів чи до інших частин мережі через синапси. Синапси встановлюють зв'язки між нейронами та передають сигнали шляхом електричних або хімічних сигналів. Вплив сигналів на синапсах може змінювати силу зв'язку між нейронами, що відображається у вагах синаптичних зв'язків. Суматори в кожному нейроні агрегують вхідні сигнали з урахуванням їх вагів і визначають, чи буде нейрон активованим. Процес передачі сигналів у нейронних мережах може бути представлений математичними моделями, такими як лінійні або нелинійні функції активації. Взаємодія між нейронами через синаптичні зв'язки формує базову структуру для обробки інформації та вирішення завдань машинного навчання. Розуміння ролі кожного з компонентів нейронної мережі є ключовим для розробки та оптимізації моделей штучного інтелекту. Далі буде розглянуто деякі основні типи архітектури побудови нейронних мереж.

1.2.1 Перцептрон

Перцептрон є однією з перших і найвідоміших моделей штучних нейронних мереж, запропонованих Френком Розенблаттом у 1957 році. Він був створений для розв'язання задач класифікації та побудови лінійно роздільних образів. Основною метою перцептрона є навчання розпізнаванню патернів через регулювання ваг зв'язків між нейронами [4].

Перцептрон складається з вхідного шару нейронів, прихованих шарів (якщо такі є) та вихідного шару нейронів. Вхідні нейрони отримують сигнали з навколишнього середовища і передають їх до наступного шару через зважені зв'язки. Ваги цих зв'язків коригуються під час процесу навчання для мінімізації помилок у виході мережі.

Основні компоненти перцептрона включають:

Вхідний шар: приймає вхідні сигнали, які є ознаками для класифікації.

Синапси: зважені зв'язки, які передають сигнали від одного нейрона до іншого.

Адаптивний суматор: обчислює скалярний добуток вхідного вектора сигналів і вектора ваг.

Активаційна функція: нелінійне перетворення вихідного сигналу суматора. Типові функції активації включають сигмоїдальну функцію, гіперболічний тангенс, лінійну і східчасту функції.

Вихідний шар: генерує кінцевий результат класифікації.

Процес навчання персептрона здійснюється за допомогою алгоритму коригування помилок [5]. Під час навчання персептрон порівнює свій вихід з бажаним виходом і коригує ваги, щоб мінімізувати різницю між ними. Це досягається за допомогою методу зворотного поширення помилки, який коригує ваги на основі градієнта функції втрат.

Перцептрон здатний вирішувати задачі лише для лінійно роздільних наборів даних. Це означає, що дані мають бути розділені на класи прямою

лінією в просторі ознак. Однак, перцептрон не може вирішувати задачі, які вимагають нелінійного поділу, наприклад, задачу XOR (виключне "або").

Для розширення можливостей перцептрона були розроблені багатошарові персептрони (MLP), які включають один або більше прихованих шарів. Ці мережі здатні розв'язувати більш складні задачі, оскільки приховані шари дозволяють моделювати нелінійні залежності між вхідними даними і вихідними результатами.

Багатошаровий перцептрон навчається також за допомогою алгоритму зворотного поширення помилки, але в цьому випадку коригування вагів здійснюється для всіх шарів мережі, що дозволяє моделі навчатися складнішим патернам і забезпечувати високу точність класифікації.

Таким чином, перцептрон є фундаментальною моделлю у сфері штучних нейронних мереж і послужив основою для подальшого розвитку більш складних нейронних мереж і методів глибокого навчання.

Існує кілька топологій, що стосуються перцептронів: подвоєний перцептрон, перехресно-подвоєний, зворотньо-подвоєний. На рисунку 1.5 зображено елементарний персептрон.

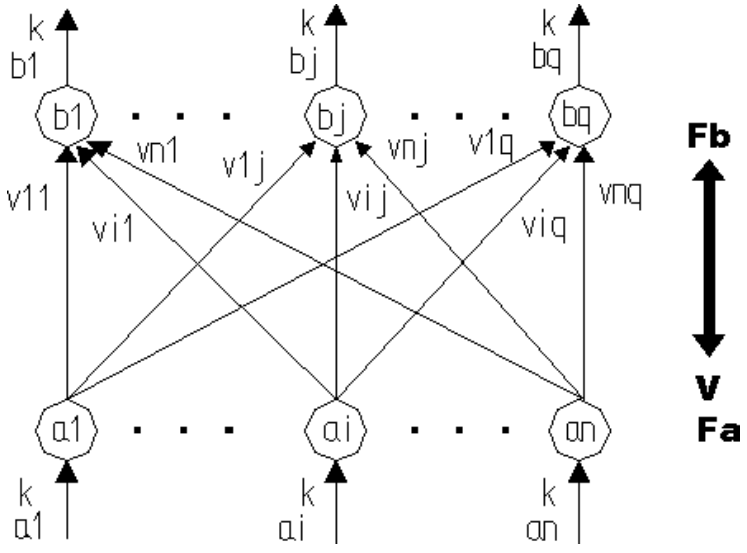

Рисунок 1.5 - Топологія односпрямованого перцептрона [4]

На рисунку 1.6 показана топологія подвоєного перцептрона – багатошарового персептрона зі статичними випадково встановленими вагами зв'язків між елементами F a та F b і вагами зв'язків, що підлаштовуються між елементами шару F b та шару F c .

Елементарний перцептрон – це двошаровий зв'язковий перетворювач наборів, який запам'ятовує зразкові пари ($A_k B_k$), $k = 1 \ldots m$., Використовуючи процедуру коригування помилок. F_a забезпечує прийом вхідних образів $A = (a_{1k}, a_{2k}, \ldots a_{nk})$, де $k = 1, 2, \ldots m$, а вихідний шар F_b складається з нейронів зі ступінчастою функцією активізації та забезпечує формування вихідних бінарних образів $B = (b_1, b_2, \ldots, b_p)$, Що приймають значення 0 і 1.

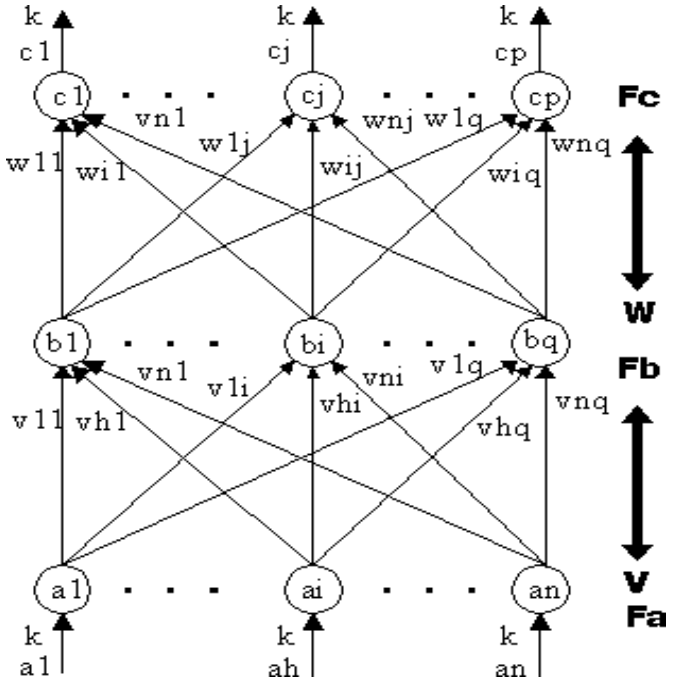

Рисунок 1.6 – Топологія подвоєного перцептрона [4]

1.2.2 Генеративно-змагальні мережі (GANs)

Генеративно-змагальні мережі (GANs) — це тип нейронних мереж, який складається з двох підмереж: генератора та дискримінатора (рисунок 1.7). Генератор створює штучні зразки даних, намагаючись зробити їх якнайближчими до реальних, тоді як дискримінатор оцінює, чи є ці зразки справжніми або

створеними генератором [6]. Цей процес змагання між генератором і дискримінатором дозволяє GANs генерувати дуже реалістичні дані після навчання.

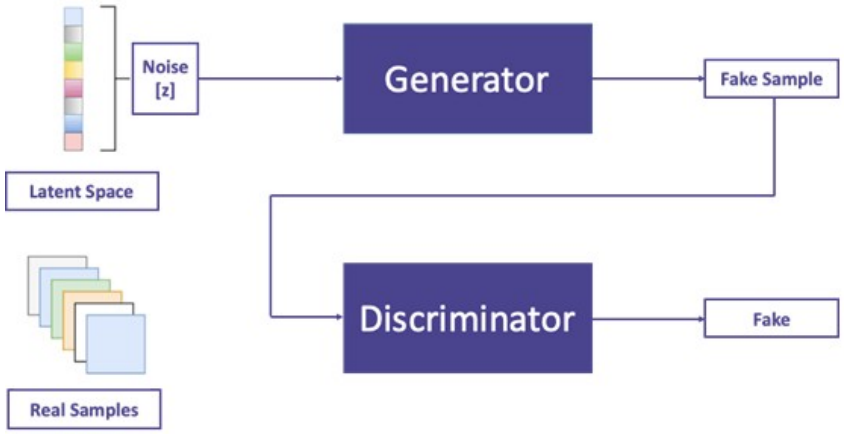

Рисунок 1.7 — Блок-схема побудови генеративно-змагальної мережі [6].

Основна ідея GANs полягає в тому, що генератор покращує свої результати, намагаючись обдурити дискримінатор, а дискримінатор вдосконалюється, намагаючись правильно визначити підроблені дані. Це створює динамічний процес, де обидві мережі постійно вдосконалюються.

GANs широко використовуються у створенні реалістичних зображень, відео, музики та інших видів даних [7]. Однією з найвідоміших архітектур GAN є Deep Convolutional GANs (DCGANs), які застосовують згорткові нейронні мережі для покращення якості генерованих зображень. Інший важливий різновид — Conditional GANs (CGANs), де генератор та дискримінатор отримують додаткову інформацію для створення умовних зразків.

Переваги GANs включають їх здатність генерувати високоякісні дані, що можна використовувати в різних застосуваннях, таких як створення віртуальної реальності, генерація нових дизайнерських рішень, та покращення якості зображень. Проте навчання GANs може бути складним через нестабільність та чутливість до налаштувань гіперпараметрів. Також існує ризик виникнення явища "mode collapse", коли генератор виробляє обмежений набір варіантів даних.

Завдяки своїй здатності створювати нові, реалістичні дані, GANs мають великий потенціал у багатьох галузях, включаючи штучний інтелект, медицину, розваги та інші сфери, що вимагають обробки та генерації даних.

1.2.3 Трансформери

Трансформери — це тип нейронних мереж, розроблений для обробки послідовних даних, особливо тексту [8]. Вони використовують механізм самоуваги (self-attention), який дозволяє моделі визначати важливі частини вхідної послідовності незалежно від їх віддаленості. Це революційна архітектура, яка значно підвищила ефективність обробки природної мови.

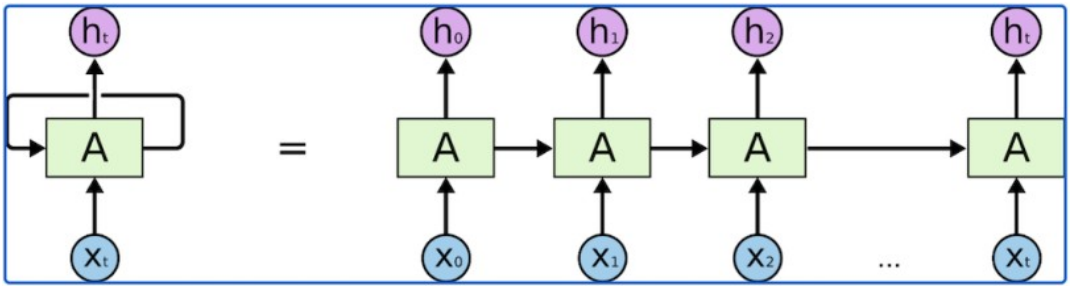

Рисунок 1.8 – Топологія мереж типу “трансформер” [8].

Однією з найвідоміших реалізацій трансформерів є модель BERT (Bidirectional Encoder Representations from Transformers), яка встановила нові стандарти для різних завдань обробки тексту. Інша популярна модель — GPT (Generative Pre-trained Transformer), яка демонструє високі результати в генерації тексту [9].

Основна перевага трансформерів полягає в їх здатності обробляти послідовності паралельно, що значно пришвидшує процес тренування у порівнянні з рекурентними нейронними мережами (RNN). Крім того, трансформери можуть зберігати довготривалі залежності у послідовностях, що робить їх ефективнішими для роботи з довгими текстами. Використання

трансформерів не обмежується лише обробкою тексту. Вони також застосовуються в задачах обробки зображень та біоінформатики. Наприклад, Vision Transformers (ViTs) використовуються для завдань комп'ютерного зору, таких як класифікація зображень і об'єктне розпізнавання [10].

Недоліком трансформерів є їх висока вимогливість до обчислювальних ресурсів. Вони потребують значних обсягів пам'яті та потужності для обробки великих обсягів даних, що може бути проблемою для менш потужних систем. Також тренування трансформерів вимагає великих наборів даних для досягнення високої точності.

Загалом, трансформери стали ключовим інструментом в сучасному машинному навчанні та обробці природної мови, відкриваючи нові можливості для дослідників та інженерів у багатьох галузях.

1.2.4 Мережі з резидуальними зв'язками (ResNet)

Мережі з резидуальними зв'язками (ResNet) представляють собою інноваційну архітектуру глибоких нейронних мереж, яка була вперше представлена у 2015 році [11]. Однією з головних переваг ResNet є використання резидуальних блоків, що дозволяють зберігати та передавати інформацію через нейронні шари. Це допомагає уникнути проблеми з втратою інформації та зникненням градієнтів у глибоких мережах. Ключовим компонентом ResNet є "скорочені з'єднання" (shortcut connections), які дозволяють обчислювальним графам пропускати деякі шари, що полегшує процес оптимізації та навчання.

Щоб зробити можливим створення дуже глибоких згорткових структур, ResNet додає проміжні вхідні дані до вихідних даних групи блоків згортки. Це також називається пропускними з'єднаннями, співставленням ідентифікаторів та «залишковими з'єднаннями». Мета пропускних з'єднань – забезпечити більш плавний потік градієнтів та збереження важливих ознак до останніх шарів. Вони

не додають обчислювального навантаження в мережу. На рисунку 1.9 показана діаграма блоку структури ResNet, де використані наступні позначення:

x — вхідні дані для блоку ResNet, що є вихідними даними попередніх шарів; F(x) — невелика нейронна мережа з кількома блоками згортки.

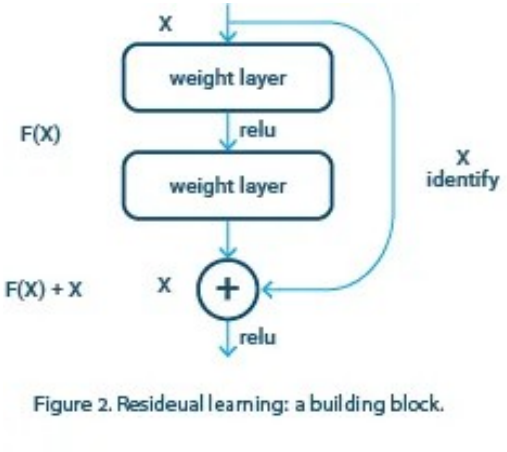

Рисунок 1.9 - Діаграма блоку структури ResNet [11]

В ResNet вхідні дані x проходять через кілька шарів згортки, утворюючи F(x), після чого x додається до результату F(x). Це виражається як y = F(x) + x, де y — це вихід блоку. Таке додавання входу напряму до виходу допомагає запобігти затуханню градієнта і покращує навчання глибокої мережі, дозволяючи їй ефективніше передавати інформацію на наступні шари.

Ця архітектура дозволяє конструювати дуже глибокі мережі (до 100 або навіть більше шарів) без виникнення проблеми з втратою продуктивності. ResNet став одним із стандартів для багатьох завдань комп'ютерного зору, таких як класифікація зображень та виявлення об'єктів [12]. Відзначається також, що ResNet є досить ефективним і надійним рішенням, яке зазвичай дає добрі результати навіть без значної тонкої настройки. Ця архітектура залишається популярною в галузі машинного навчання та глибокого навчання на сьогодні.

1.2.5Довга короткочасна пам'ять (LSTM)

Long Short-Term Memory (LSTM) є різновидом рекурентних нейронних мереж (RNN), призначених для обробки послідовних даних та подолання проблеми зникання градієнта. LSTM складається з блоків пам'яті, кожен з яких контролюється трьома воротами: входу, забуття та виходу [13]. Вхідні ворота визначають, яка нова інформація повинна бути збережена, ворота забуття вирішують, яку частину попереднього стану слід відкинути, а вихідні ворота контролюють, яка частина стану блоку буде використана для формування виходу. Структура мережі показана на рисунку 1.10.

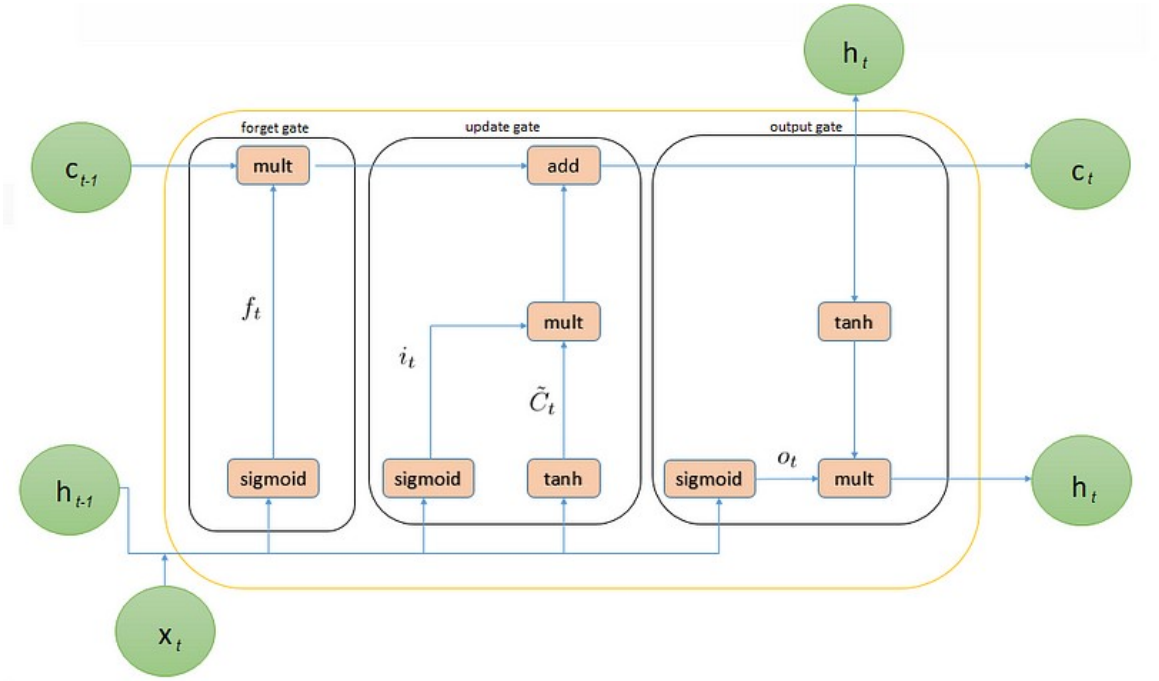

Рисунок 1.10 Структура мережі LSTM [13].

Основна перевага LSTM полягає в їх здатності зберігати довготривалі залежності, що робить їх ефективними для завдань обробки природної мови, розпізнавання мовлення та прогнозування часових рядів. Наприклад, у задачах машинного перекладу LSTM можуть враховувати контекст всього речення, що покращує точність перекладу. Вони також використовуються в фінансових

моделях для прогнозування ринкових трендів та в медицині для аналізу послідовних даних пацієнтів [14].

Незважаючи на їх переваги, LSTM мають деякі недоліки, включаючи високу обчислювальну складність та тривалий час навчання. Через їх складну архітектуру, моделі LSTM можуть потребувати більше даних для навчання та схильні до перенавчання, якщо не використовувати регуляризацію.

Загалом, LSTM мережі є потужним інструментом для обробки послідовних даних, забезпечуючи збереження та використання довготривалих залежностей, що відкриває нові можливості в багатьох галузях.

2. ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ АЛГОРИТМУ РОЗПІЗНАВАННЯ МОВЛЕННЯ

На рисунку 2.1 показана електрична принципова схема експериментального макету для перевірки алгоритму розпізнавання мовлення.

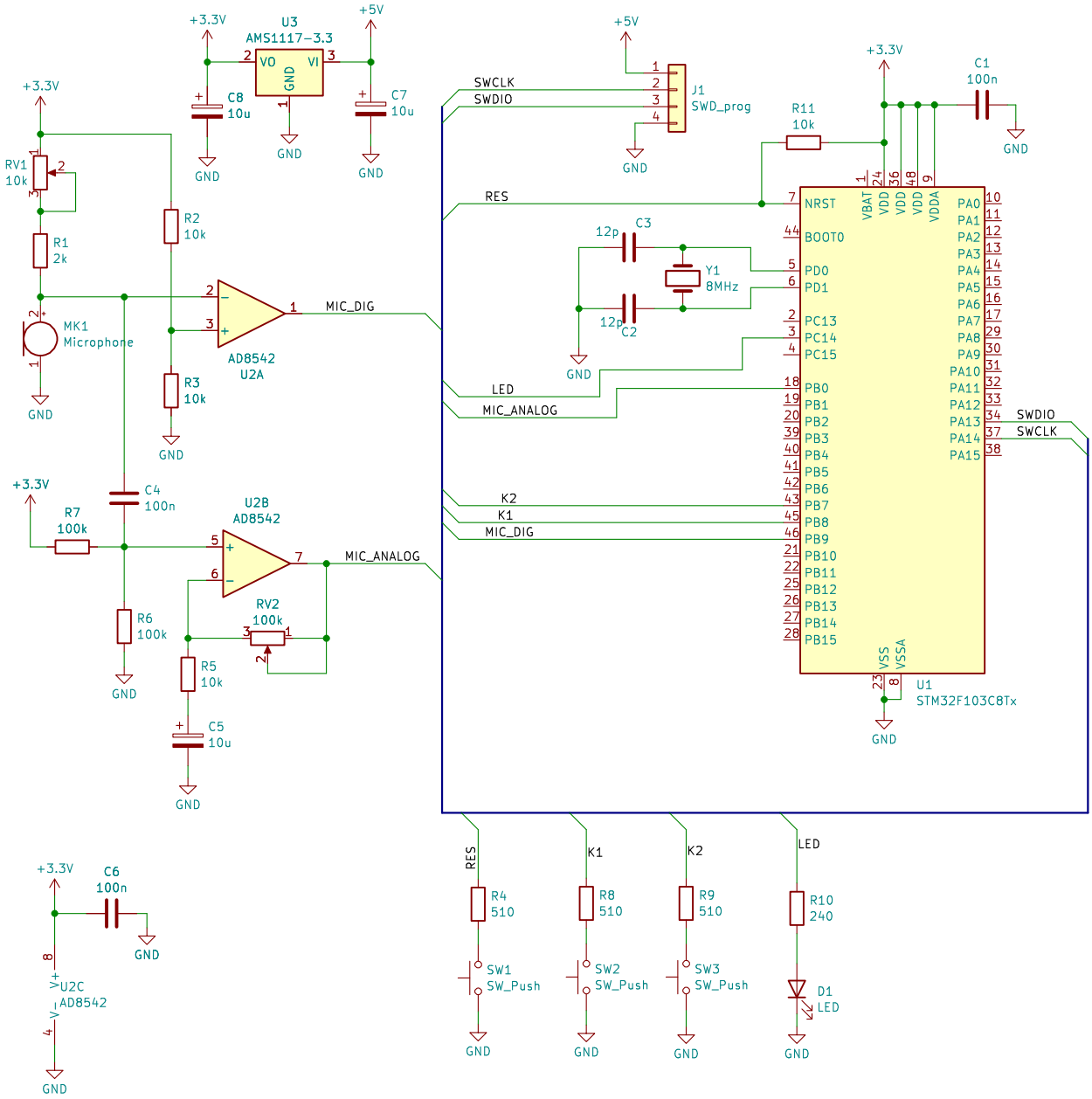

Рис. 2.1 Принципова електрична схема експериментального макету системи розпізнавання мовлення.

Основу схеми складає мікроконтролер STM32F103C8T6, який побудований за архітектурою ARM (ядро CORTEX-M3) та працює на частоті кварцового резонатора 8 МГц із множником частоти 9, що відповідає його максимальній частоті 72 МГц [15]. Аналізований звуковий фрейм перетворюється електретним мікрофоном МК1 (CZN-15E) у електричний сигнал, який підсилюється операційним підсилювачем U2B і подається на вхід АЦП мікроконтролера. Використаний операційний підсилювач AD8542 типу rail-to-rail дозволяє використовувати однополярне живлення. Підсилювач сигналу мікрофона побудований за схемою неінвертуючого увімкнення ОП з можливістю регулювання коефіцієнта підсилення змінним резистором RV2 в діапазоні від 1 до 11. Конденсатор C5 доданий у коло зворотнього зв'язку забезпечує одиничний коефіцієнт передачі підсилювача для постійної складової. Тому, постійна складова сигналу на неінвертуючому вході ОП утворена резисторами R6 та R7 (половина напруги живлення) створює таку ж постійну складову на виході ОП — незалежно від положення регулювального гвинта змінного резистора RV2. Наявність постійної складової у сигналі, який подається на АЦП мікроконтролера необхідна для можливості оцифровки двополярного сигналу мікрофона з однополярним живленням АЦП. Постійна складова напруги на неінвертуючому вході ОП U2B відділяється від постійної складової мікрофона розділовим конденсатором C4.

Для забезпечення робочої точки мікрофона на нього подається напруга живлення через послідовно включені постійний R1 та змінний RV1 резистори. Постійний резистор необхідний для обмеження струму у випадку встановлення нульового значення змінного резистора. Сигнал мікрофона разом із постійною складовою, яка відповідає його робочій точці подається не лише на підсилювач U2B, а й на інвертуючий вхід ОП U2A, який працює в режимі компаратора. На неінвентуючий вхід ОП U2A подається половина напруги живлення з подільника побудованого на резисторах R2 та R3. У процесі налаштування опір резистора

RV1 встановлюється таким чином, щоб постійна складова напруги мікрофона була дещо нижчою від половини напруги живлення, отже на виході ОП U2A встановлюється високий рівень напруги. Якщо мікрофон почне сприймати значущий по амплітуді сигнал, то напруга на його інвертуючому вході зазнає коливань, піки яких будуть перевищувати порогове значення половини напруги живлення і на виході ОП U2A почнуть з’являтись імпульси низького рівня напруги. Цей сигнал подається на 9 пін порту B мікроконтролера і за ним він визначає, що відбувається передача фрази. Слід зауважити, що використання операційного підсилювача в якості компаратора не завжди добре через можливу неузгодженість вихідної напруги із логічними рівнями цифрової схеми, до якої він підключається та підвищеними затримками через вхід у насичення внутрішніх каскадів ОП при ненульовій різниці напруги між інвертуючим та неінвертуючим входами ОП. Однак, у нашому випадку використаний ОП типу rail-to-rail, напруга живлення якого така ж, як і мікроконтролера, тому буде повна узгодженість цифрових рівнів. Щодо затримок, то у даному випадку їх вплив некритичний, оскільки розглядаються порівняно повільні процеси.

Для програмування мікроконтролера використовується програматор ST-LINK V2, який підключається до SWD інтерфейсу через роз’єм J1. Живлення схеми також отримується через роз’єм J1. Хоча програматор ST-LINK V2 має вбудований стабілізатор напруги 3.3В, у схемі застосований окремий стабілізатор на IMC AMS1117-3.3, що зберігає стабілізатор програматора від перевантаження.

Кнопки SW2 та SW3 використовуються для переведення програмного алгоритму у режим навчання одній із двох фраз. Для індикації результату розпізнавання фрази використовується світлодіод D1.

На рисунку 2.2 показана блок схема основного циклу алгоритму розпізнавання мовлення, який реалізований на основі спектрального аналізу, як і в роботі [1]. При кожному проході відбувається перевірка стану кнопок SW2 та SW3, якщо якась із них натиснута, то відбувається перехід до процесу навчання

відповідній фразі. Якщо ж обидві кнопки відпущені, то переходимо до блоку розпізнавання фрази, якщо така фраза була прийнята. Слід зауважити, що код містить два однакові блоки коду навчання для кнопок SW2 та SW3, які відповідають двом різним фразам.

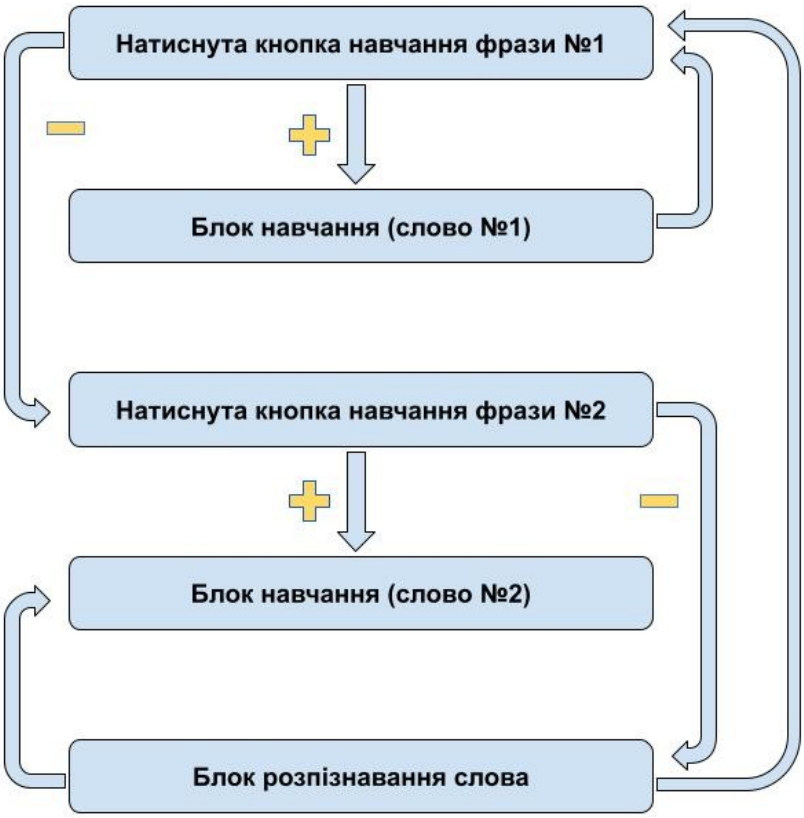

Рисунок 2.2 - Блок схема основного циклу алгоритму розпізнавання мовлення.

Одна з таких фраз має засвітити світлодіод індикації, а інша — погасити його. Звісно, реалізований алгоритм не обмежується двома фразами, але у даному випадку метою було зробити не закінчений пристрій, а лише макет для перевірки можливостей алгоритму.

На рисунку 2.3 показана блок-схема алгоритму навчання фрази, вхід до якого як було показано вище виконується натисканням відповідної кнопки. У цьому блоці алгоритм чекає на цифровий сигнал мікрофона (пін PB9), низький рівень якого сигналізує про наявність сигналу значущої амплітуди на вході мікрофона. Коли такий сигнал з’являється запускається процес оцифровки

аналогового сигналу мікрофона. Тоді, дані з АЦП зі сталою частотою дискретизації записуються у буфер сталої довжини, який реалізований механізмом прямого доступу до пам’яті (DMA).

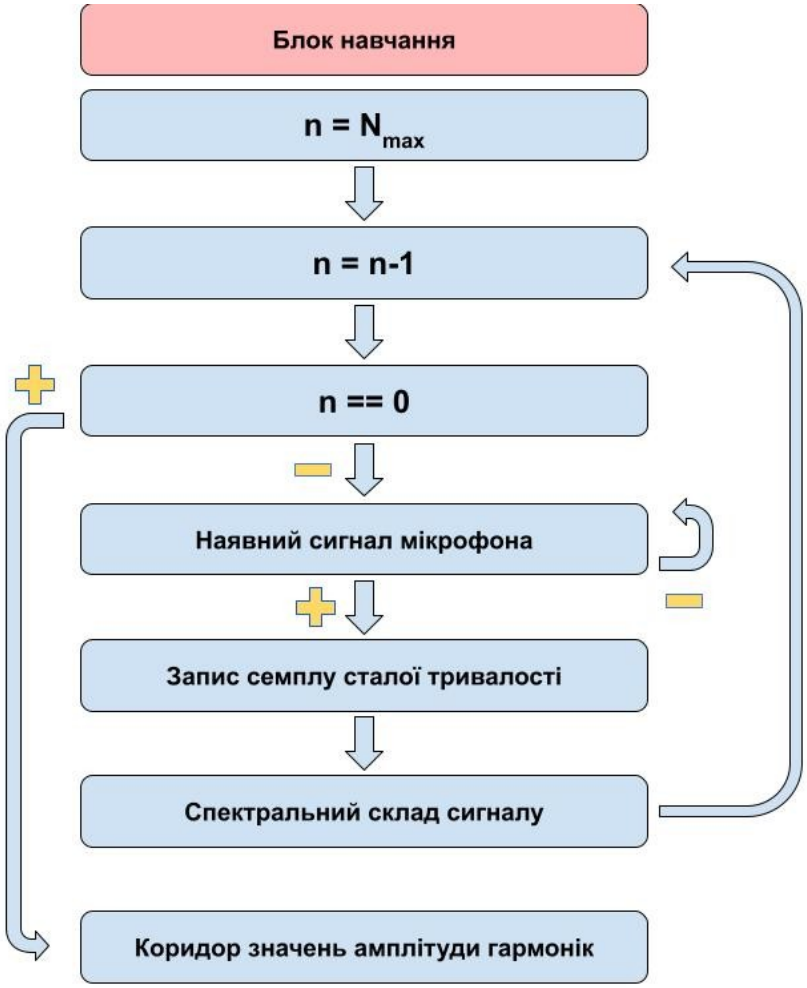

Рисунок 2.3 - Блок-схема алгоритму навчання фрази.

Розмір буфера у нашому випадку становив 256 байт. Після заповнення буфера реалізується алгоритм швидкого перетворення Фур’є, на виході якого отримується спектральний склад сигналу. З цього спектрального складу обирається фіксована кількість гармонік (починаючи з першої), яка задається у коді до компіляції. Амплітуди цих обраних гармонік заносяться до окремого буфера, який відповідає окремій фразі з номером n. Далі, алгоритм чекає на прийом наступної такої ж фрази і записує таку ж обмежену кількість гармонік до

буфера, який відповідає вже другій фразі і так відбувається *Nmax* разів. Тобто, алгоритм навчання складається з прийому *Nmax* однакових фраз і запису обмеженої кількості їх перших гармонік до окремих буферів з подальшим визначенням максимальної та мінімальної амплітуди у наборі для кожної гармоніки. Ось саме цей коридор допустимих значень кожної гармоніки і є результатом навчання. У нашому *Nmax* = 10, тобто під час навчання треба було повторити одну і ту ж фразу 10 разів. Слід зауважити, що нульова гармоніка (постійна складова) не береться до уваги у процесі навчання, і як далі побачимо у процесі розпізнавання, тому додавання постійної складової до вихідної напруги ОП U2B не вплинуло на результат розпізнавання.

На рисунку 2.4 показана блок-схема алгоритму розпізнавання мовлення. Тут, алгоритм так само, як і в процесі навчання очікує на наявність сигналу мікрофона, про який свідчить низький рівень на піні PB9.

Рисунок 2.4 - Блок-схема алгоритму розпізнавання мовлення.

У випадку наявності сигналу, запускається процес оцифровки сигналу з тією ж частотою дискретизації та розміром буферу, що і в процесі навчання. Потім, розраховується спектральний склад записаного сигналу за допомогою швидкого перетворення Фур'є. З цього складу обирається обмежена кількість гармонік, як і в процесі навчання і відбувається перевірка, чи вони потрапляють у коридор амплітуд гармонік, який був визначений у процесі навчання. Якщо всі обрані гармоніки потрапляють у визначений для якоїсь фрази коридор, то вважаємо, що фраза розпізнана, і робиться відповідна дія (у нашому випадку засвічуємо або гасимо світлодіод).

Повний програмний код з детальними коментарями наведений у додатку.

На рисунку 2.5 показаний зовнішній вигляд експериментального макету, який виконаний за допомогою макетної плати.

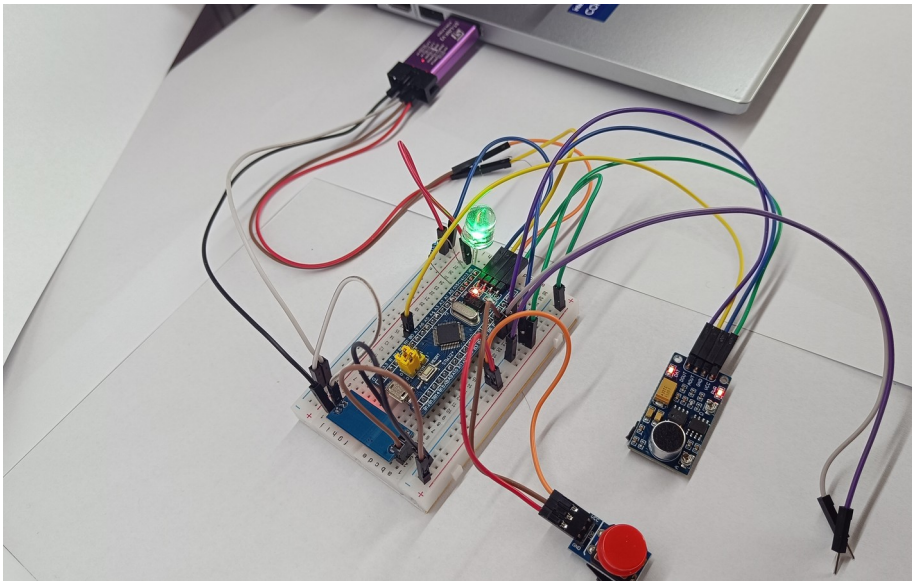

Рисунок 2.5 - Зовнішній вигляд експериментального макету.

Аналіз роботи алгоритму показав, достатню ефективність розпізнавання для його використовувати у побудові систем голосового керування. Однак, інколи, доводиться повторювати одну і ту ж фразу. Імовірно, підбираючи тривалість запису фрейму і кількість гармонік розпізнавання можна досягнути майже 100% ефективності розпізнавання.

ВИСНОВКИ

В даній роботі на основі мікроконтролера STM32F103C8T6 (CORTEX-M3) реалізовано алгоритм розпізнавання мовлення, який належить до загальної задачі розпізнавання образів. З множини можливих підходів було обрано спектральний алгоритм, який має достатню точність та не надто високу складність. Його суть полягає у порівнянні спектру фрази, яка розпізнається із спектром опорної фрази, який отриманий у процесі навчання. Реалізований алгоритм показав достатньо високу ефективність розпізнавання, що дозволяє його використовувати у задачах побудови систем із голосовим керуванням.

Однак, запропонований алгоритм не дозволяє виокремлювати фразу із потоку мовлення, для нього необхідна окрема голосова команда. Також, імовірно, наявність значного рівня шуму може зашкодити успішному розпізнаванню команди.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. О.Ю.Белобородов, Розпізнавання аудіосигналів з використанням обертонового ряду, Національный аерокосмічний університет ім. Н.Е.Жуковского «ХАІ»
2. Бохан К.О., Кучук Г.А. Методи цифрової обробки сигналів: Навчальний посібник для студентів комп'ютерних спеціальностей вищих навчальних закладів. – Харків: Національний аерокосмічний університет ім. М.Є. Жуковського „Харківський авіаційний інститут”, 2008 – 84 с.
3. https://pubs.aip.org/aip/apl/article/122/7/074102/2876778/CMOS-based-area-and-power-efficient-neuron-and
4. Проф. Ковалевський с. ст., конспект лекцій по курсу «Основи сучасних теорій моделювання процесів», Краматорськ 2018
5. https://www.geeksforgeeks.org/backpropagation-in-neural-network/ (дата звернення: 17.05.2024).
6. https://neptune.ai/blog/6-gan-architectures (дата звернення: 17.05.2024).
7. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (дата звернення: 17.05.2024).
8. https://builtin.com/artificial-intelligence/transformer-neural-network (дата звернення: 17.05.2024).
9. https://deeprevision.github.io/posts/001-transformer (дата звернення: 17.05.2024).
10. https://datascientest.com/en/transformer-neural-network-what-is-it-how-does-it-work (дата звернення: 17.05.2024).
11. https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet (дата звернення: 18.05.2024).
12. https://builtin.com/artificial-intelligence/resnet-architecture (дата звернення: 18.05.2024).

13. https://towardsai.net/p/machine-learning/demystifying-the-architecture-of-long-short-term-memory-lstm-networks (дата звернення: 18.05.2024).
14. https://d2l.ai/chapter_recurrent-modern/lstm.html (дата звернення 18.05.2024).
15. https://www.st.com/resource/en/datasheet/stm32f103c8.pdf (дата звернення: 19.05.2024).

ДОДАТОК А

```
#include "stm32f10x.h"   // стандартний заголовковий файл
#include <math.h>
#define true 1        // літерні назви булевих величин
#define false 0
typedef unsigned char bool;       // оголошуємо тип даних bool
#define ADCSIZERX     256    // розмір буфера під ацп дані
#define        HARMONIC_SIZE        10      // кількість гармонік навчання
#define HARM_NUMBER       5       // кількість гармонік для розпізнавання
volatile static uint16_t ADCBufferRx[ADCSIZERX];      //буфер під adc1-дма
static double ADCBufferRxFourier[ADCSIZERX]; // буфер під фур'є гармоніки
void DMA1_Channel1_IRQHandler(void);  //дма 1 канал 1 прийом даних з ацп
static double dHarm[HARM_NUMBER][HARMONIC_SIZE];
static uint8_t u8HarmonicValueCounter = 0;
static double dStartWordHarm[HARM_NUMBER][2];
static double dStopWordHarm[HARM_NUMBER][2];
// функції для пошуку мінімального та максимального значення в масиві
static double Min(double dData[HARM_NUMBER][HARMONIC_SIZE], uint8_t
n_harm, uint16_t u16Size)
{
double dMin = dData[n_harm][0];
for(uint16_t i=0; i<u16Size; i++)
if(dData[n_harm][i] < dMin)
dMin = dData[n_harm][i];
return dMin;
}
```

```
static double Max(double dData[HARM_NUMBER][HARMONIC_SIZE], uint8_t
n_harm, uint16_t u16Size)
{
double dMax = dData[n_harm][0];
for(uint16_t i=0; i<u16Size; i++)
if(dData[n_harm][i] > dMax)
dMax = dData[n_harm][i];
return dMax;
}
// Перетворення Фур'є
// AVal - масив аналізованих даних, Nvl - довжина масиву повинна бути кратна
ступеня 2
// FTvl - масив отриманих значень, Nft - довжина масиву повинна дорівнювати
Nvl.
static const double TwoPi = 6.283185307179586;
static double Tmvl[ADCSIZERX*2];
static void FFTAnalysis(const double *AVal, double *FTvl, int Nvl, int Nft) {
 int i, j, n, m, Mmax, Istp;
  double Tmpr, Tmpi, Wtmp, Theta;
 double Wpr, Wpi, Wr, Wi;
//double *Tmvl;
 n = Nvl * 2; //Tmvl = new double[n];   Tmvl перед функцией
 for (i = 0; i < n; i+=2) {
 Tmvl[i] = 0;
 Tmvl[i+1] = AVal[i/2];
}
i = 1; j = 1;
while (i < n) {
```

```
if (j > i) {
 Tmpr = Tmvl[i]; Tmvl[i] = Tmvl[j]; Tmvl[j] = Tmpr;
Tmpr = Tmvl[i+1]; Tmvl[i+1] = Tmvl[j+1]; Tmvl[j+1] = Tmpr;
 }
 i = i + 2; m = Nvl;
while ((m >= 2) && (j > m)) {
 j = j - m; m = m >> 1;
 }
 j = j + m;
}
 Mmax = 2;
while (n > Mmax) {
Theta = -TwoPi / Mmax; Wpi = sin(Theta);
 Wtmp = sin(Theta / 2); Wpr = Wtmp * Wtmp * 2;
 Istp = Mmax * 2; Wr = 1; Wi = 0; m = 1;
 while (m < Mmax) {
 i = m; m = m + 2; Tmpr = Wr; Tmpi = Wi;
Wr = Wr - Tmpr * Wpr - Tmpi * Wpi;
Wi = Wi + Tmpr * Wpi - Tmpi * Wpr;
 while (i < n) {
 j = i + Mmax;
 Tmpr = Wr * Tmvl[j] - Wi * Tmvl[j-1];
Tmpi = Wi * Tmvl[j] + Wr * Tmvl[j-1];
Tmvl[j] = Tmvl[i] - Tmpr; Tmvl[j-1] = Tmvl[i-1] - Tmpi;
 Tmvl[i] = Tmvl[i] + Tmpr; Tmvl[i-1] = Tmvl[i-1] + Tmpi;
i = i + Istp;
}
}
```

```
 Mmax = Istp;
}
for (i = 0; i < Nft; i++) {
 j = i * 2; FTvl[i] = 2*sqrt(pow(Tmvl[j],2) + pow(Tmvl[j+1],2))/Nvl;
}
//delete []Tmvl;
}
int fso;
int main(void)
{
//---SETUP RCC
RCC_DeInit();       // скид налаштувань тактового генератора
RCC_HSICmd(DISABLE); //вимкнення внутрішнього RC HSI 8МГц генератора
RCC_HSEConfig(RCC_HSE_ON);    // включення зовнішнього кварцу
RCC_PLLConfig(RCC_PLLSource_HSE_Div1 ,RCC_PLLMul_9);
//  тактування від HSE з PREDIV1, дільник частоти кварцу = 1, множник = 9 отри-
муємо (8/1) * (9) = 72МГц
RCC_PLLCmd(ENABLE);       //Вмикаємо PLL
RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);         // На системну шину
подаємо тактування з множника PLLMUL
RCC_PCLK1Config(RCC_HCLK_Div2);    //частота PLCK1 (APB1)= (SYSCLK) / 2
= 36 МГц
RCC_PCLK2Config(RCC_HCLK_Div1);    //частота PLCK2 (APB2)= (SYSCLK) / 1
= 72 МГц
RCC_ADCCLKConfig(RCC_PCLK2_Div6);         //частота ADCCLK = PLCK2/6 =
72 / 6 = 12 МГц
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE)
//---SETUP GPIO
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;     // світлодіод pc.13
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC, &GPIO_InitStructure);
//---мікрофон аналоговий
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
//---управління налаштувальним світлодіодом
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14;     //Зовнішній світлодіод
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//push-pull (два рівня 0 и 1, немає прив'язки через внутрішнього  резистора до жи-
влення)
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC, &GPIO_InitStructure);
//---мікрофон цифровий
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;       //мікрофон цифровий pb.9
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;       // кнопка навчання 1 pb.7
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;// кнопка навчання 2 pb.8
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);
//---SETUP TIMER
TIM_TimeBaseInitTypeDef TIMER_InitStructure;
TIMER_InitStructure.TIM_CounterMode = TIM_CounterMode_CenterAligned3;  //
рахуємо вгору і вниз, переривання, коли 0 і коли досягли регістру ARR
TIMER_InitStructure.TIM_Prescaler = 720-1;// дільник, 10 мкс відлік
TIMER_InitStructure.TIM_Period = 10-1;
TIMER_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIMER_InitStructure.TIM_RepetitionCounter = 1-1;
TIM_TimeBaseInit(TIM1, &TIMER_InitStructure);
        //---SETUP DMA
DMA_InitTypeDef DMA_InitStructure;       //структура опису канала dma
//1й канал - це ADC
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)(&(ADC1->DR));
//зв’язуємо с adc1 регістр даних DR
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) ADCBufferRx;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = ADCSIZERX;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
        DMA_InitStructure.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;
```

```
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
// збільшення адреси периферії, у нас не змінюємо
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
// нормальна передача без зациклювання
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_Priority = DMA_Priority_High; // пріоритет високий
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
//---SETUP ADC
ADC_InitTypeDef ADC_InitStructure;
ADC_StructInit(&ADC_InitStructure);
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T3_TRGO;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(ADC1, &ADC_InitStructure);
ADC_RegularChannelConfig(ADC1,ADC_Channel_8,1,ADC_SampleTime_71Cycles
5);
ADC_Cmd(ADC1,ENABLE);
ADC_ResetCalibration(ADC1);
 АЦПwhile(ADC_GetResetCalibrationStatus(ADC1));
ADC_StartCalibration(ADC1);
while(ADC_GetCalibrationStatus(ADC1));
ADC_Cmd(ADC1,ENABLE);  // перезапускаємо АЦП
ADC_DMACmd(ADC1, ENABLE);
        //---SETUP NVIC
```

```
TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);// вхід у переривання таймера
NVIC_EnableIRQ(TIM1_UP_IRQn); // вмикаємо переривання для таймера 1
NVIC_SetPriority(TIM1_UP_IRQn,1);//пріоритет=1
DMA_ITConfig(DMA1_Channel1,DMA_IT_TC,ENABLE);
NVIC_EnableIRQ(DMA1_Channel1_IRQn);
NVIC_SetPriority(DMA1_Channel1_IRQn,0);
        //---START WORK
double A_first_harm; // амплітуда першої гармоніки
while(true)
{
TIM_Cmd(TIM1,ENABLE);
 if(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_7) == 0) // якщо є сигнал із кнопки
навчання 1
{
u8HarmonicValueCounter = 0;  //обнуляємо лічільник значень гармонік
//цикл навчання
while(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_7) == 0)
// приймаємо голос, поки кнопка натиснута
{
if(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_9))
// якщо є сигнал із цифрового виходу мікрофона
{
DMA_SetCurrDataCounter(DMA1_Channel1, ADCSIZERX);  // установка в дма
скільки даних читати з ацп
DMA_Cmd(DMA1_Channel1, ENABLE);
//вмикаємо канал
while((DMA1_Channel1->CCR & 0x01) == SET); // чекаємо доки він працює, канал
сам вимкнеться в перериванні DMA1_Channel1_IRQHandler
```

```
for(uint16_t i=0; i<ADCSIZERX; i++)
ADCBufferRxFourier[i] = ((double) ADCBufferRx[i])*3.3/4096;
// робимо Фур'є перетворення
FFTAnalysis(ADCBufferRxFourier,ADCBufferRxFourier,ADCSIZERX,ADCSIZERX
);
// копіюємо дані фур'є гармонік в масиви окремих гармонік dHarm
A_first_harm = ADCBufferRxFourier[1];
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
dHarm[k][u8HarmonicValueCounter] = ADCBufferRxFourier[k+1] / A_first_harm;
}
u8HarmonicValueCounter++;
if(u8HarmonicValueCounter>=HARMONIC_SIZE)
u8HarmonicValueCounter = 0;
for(volatile long i=0;i<500000;i++) {GPIO_SetBits(GPIOB, GPIO_Pin_2);}
GPIO_ResetBits(GPIOB, GPIO_Pin_2);
}
}
// користувач відпустив кнопку навчання 1
// шукаємо мінімум і максимум у масивах, встановлюємо в масиви, що належать
до слів
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
dStartWordHarm[k][0] = Min(dHarm, k, HARMONIC_SIZE);  dStartWordHarm[k][1]
= Max(dHarm, k, HARMONIC_SIZE);
}
continue;}
if(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_8) == 0)  // якщо є сигнал із кнопки 2
```

```
{
u8HarmonicValueCounter = 0;
//цикл навчання
while(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_8) == 0)      // приймаємо голос, поки кнопка натиснута
{
if(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_9)) якщо є сигнал із цифрового виходу мікрофона
{
DMA_SetCurrDataCounter(DMA1_Channel1, ADCSIZERX);  // установка в дма скільки даних читати з ацп
DMA_Cmd(DMA1_Channel1, ENABLE);   //вмикаємо канал
while((DMA1_Channel1->CCR & 0x01) == SET); // чекаємо доки він працює, канал сам вимкнеться в перериванні DMA1_Channel1_IRQHandler
for(uint16_t i=0; i<ADCSIZERX; i++)
ADCBufferRxFourier[i] = ((double) ADCBufferRx[i])*3.3/4096;
// робимо Фур'є перетворення
FFTAnalysis(ADCBufferRxFourier,ADCBufferRxFourier,ADCSIZERX,ADCSIZERX);
A_first_harm = ADCBufferRxFourier[1];
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
dHarm[k][u8HarmonicValueCounter] = ADCBufferRxFourier[k+1] / A_first_harm;
}
u8HarmonicValueCounter++;
if(u8HarmonicValueCounter>=HARMONIC_SIZE)
u8HarmonicValueCounter = 0;  for(volatile long i=0;i<500000;i++) {}// за цей час цифровий вихід мікрофона встановиться в логічний 0
```

```
}
}
// користувач відпустив кнопку навчання 2
// шукаємо мінімум і максимум у масивах, встановлюємо в масиви, що належать до слів
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
dStopWordHarm[k][0] = Min(dHarm, k ,HARMONIC_SIZE);  dStopWordHarm[k][1] = Max(dHarm, k ,HARMONIC_SIZE);
}
// навчання закінчено
continue;
}
if(GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_9)) // якщо є сигнал із цифрового виходу мікрофона
{
DMA_SetCurrDataCounter(DMA1_Channel1, ADCSIZERX);  // установка в дма скільки даних читати з ацп
DMA_Cmd(DMA1_Channel1, ENABLE);   //вмикаємо канал
while((DMA1_Channel1->CCR & 0x01) == SET);          // чекаємо доки він працює, канал сам вимкнеться в перериванні DMA1_Channel1_IRQHandler
for(uint16_t i=0; i<ADCSIZERX; i++)
ADCBufferRxFourier[i] = ((double) ADCBufferRx[i])*3.3/4096;
        // робимо Фур'є перетворення
FFTAnalysis(ADCBufferRxFourier,ADCBufferRxFourier,
        ADCSIZERX,ADCSIZERX);
// аналізуємо масив із фур'є гармонікам
//ADCBufferRxFourier[X] - значення гармоніки X
```

```
//dStartWordHarmX[Y] - начення для слова "1" гармоніки X мінімуму (Y=0) або максимуму (Y=1)
uint8_t temp;
temp = 1;
A_first_harm = ADCBufferRxFourier[1];
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
ADCBufferRxFourier[k+1] = ADCBufferRxFourier[k+1] / A_first_harm;
if(	(ADCBufferRxFourier[k+1] < dStartWordHarm[k][0])	||
(ADCBufferRxFourier[k+1] > dStartWordHarm[k][1]) )
{
temp = 0;
}
}
if(	temp == 1){
GPIO_SetBits(GPIOC, GPIO_Pin_13); //засвічуємо світлодіод
continue;
}
temp = 1;
for(uint8_t k=0; k < HARM_NUMBER; k++)
{
if(	(ADCBufferRxFourier[k+1] < dStopWordHarm[k][0])	||
(ADCBufferRxFourier[k+1] > dStopWordHarm[k][1]) )
{
temp = 0;
}
if(	temp == 1)
{
```

```
GPIO_ResetBits(GPIOC, GPIO_Pin_13);      //гасимо світлодіод
continue;
}
}
}
}
// переривання від таймера 1
void TIM1_UP_IRQHandler(void)
{
if(fso == 0)
{
fso = 1;
GPIO_SetBits(GPIOC, GPIO_Pin_14);
}
else{
fso = 0;
GPIO_ResetBits(GPIOC, GPIO_Pin_14);
}
if(TIM_GetITStatus(TIM1, TIM_IT_Update) == SET)     // перевірка чи встановлено прапорець, що таймер оновився
{
TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
// очищаємо прапор, щоб не зайти сюди відразу після виходу
}
}
// обробник переривання закінчення прийому даних по дма для spi1
void DMA1_Channel1_IRQHandler(void)
{
```

```
if(DMA_GetITStatus(DMA1_IT_TC1) == SET)    // перевірка прапорця входу у
переривання
 {
DMA_ClearITPendingBit(DMA1_IT_TC1);        // очищуємо прапорець входу
DMA_Cmd(DMA1_Channel1, DISABLE);           // вимикаємо дма канал
}
}
```