НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32

Виконав (-ла): студент (-ка) 4 курсу, групи ІВ-81
(шифр групи)

Яшан Оксана Володимирівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Таранюк В.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант проф. д.т.н. Сімоненко В. П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

(підпис)

“__” _____2022 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Яшан Оксани Володимирівни

1. Тема проєкту *Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32*

керівник проєкту ассистент Таранюк В.А.,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____ 2022_ року №_____

2. Термін здачі студентом закінченого проєкту *25 травня 2022 р.*

3. Вихідні дані до проєкту *технічна документація для STM32F4DISCOVERY, статичні та теоретичні дані*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

_1) Аналіз вимог до програмного та системного забезпечення: основні терміни, порівняння існуючих продуктів та встановлення технічних вимог

_2) Огляд технологій для проектування інформаційних систем управління

_3) Розробка та опис архітектури системи

_4) Тестування для системи та інструкція для користувача

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема, функціональна схема, принципова схема.

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормо-контроль	Сімоненко В. П. проф.		

7. Дата видачі завдання_____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2022-25.03.2022*	
4.	*Розробка структур окремих підсистем*	*25.03.2022-05.04.2022*	
5.	*Програмна реалізація системи*	*05.04.2022-15.04.2022*	
6.	*Оформлення пояснювальної записки*	*15.04.2022-20.05.2022*	
7.	*Захист програмного продукту*	*23.05.2022*	
8.	*Передзахист*	*29.05.2022*	
9.	*Захист*	*25.06.2022*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

В бакалаврському дипломному проєкті реалізовано навчальний програмно-апаратний комплекс для роботи з клавіатурою 4x4 в STM32.

Програма надає можливість здійснення вводу даних з клавіатури 4x4 та обробку введених даних для подальшої передачі даних для виводу на дисплей або на термінал ПК через віртуальний COM порт. Програмний продукт створений на мові С у інтегрованому візуальному середовищі STM32CubeIDE 1.7.0. Для відлагодження роботи програми використано вбудований засіб ST-LINK – програматор та відлагоджувач.

Для відтворення програми окрім плати STM32F4DISCOVERY, необхідні клавіатури 4x4 та диплей WH1602.

Annotation

In this project for a Bachelor's Degree, educational software and hardware complex project for interaction with the 4x4 keyboard in STM32 is realized.

The program provides ability to enter data from a 4x4 keyboard and process the entered data for further data transfer for display or to a PC terminal via a virtual COM port. The software product is created in C language in the integrated visual environment STM32CubeIDE 1.7.0. To debug the program, the built-in ST-LINK tool - programmer and debugger was used.

For running the program in addition to the STM32F4DISCOVERY board, you will need 4x4 keyboards and WH1602 display.

ОПИС АЛЬБОМУ

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра
	А4	***ІАЛЦ.467200.001 ОА***	Навчальний програмно-апаратний	1	
			комплекс для роботи з клавіатурою в STM32		
			Опис альбому		
	А4	***ІАЛЦ.467200.002 ТЗ***	Навчальний програмно-апаратний	3	
			комплекс для роботи з клавіатурою в STM32		
			Технічне завдання		
	А4	***ІАЛЦ.467200.003 ПЗ***	Навчальний програмно-апаратний	84	
			комплекс для роботи з клавіатурою в STM32		
			Пояснювальна записка		
	А4	***ІАЛЦ.467200.004 Д1***	Навчальний програмно-апаратний	1	
			комплекс для роботи з клавіатурою в STM32		
			Структурна схема		
	А4	***ІАЛЦ.467200.005 Д2***	Навчальний програмно-апаратний	1	
			комплекс для роботи з клавіатурою в STM32		
			Функціональна схема		
	А4	***ІАЛЦ.467200.006 Д3***	Навчальний програмно-апаратний	1	
			комплекс для роботи з клавіатурою в STM32		
			Принципова схема		
	А4	***ІАЛЦ.467200.007 Д4***	Навчальний програмно-апаратний	1	
			комплекс для роботи з клавіатурою в STM32		
			Алгоритм імплементації		
	А4	***ІАЛЦ.467200.008 Д5***	Навчальний програмно-апаратний	19	
			комплекс для роботи з клавіатурою в STM32		
			Текст програми		

					ІАЛЦ.467200.001 ОА
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розроб.		*Яшан О.В.*			*Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32 Опис альбому*
Перевір.		*Таранюк В.А.*			
Н. контр.		*Сімоненко В.П.*			
Затв.		*Стіренко С.Г.*			

Літ.	*Арк.*	*Аркушів*
	1	*1*

КПІ ім. Ігоря Сікорського
ФІОТ ІВ-81

Технічне завдання

до дипломного проєкту
на тему: «Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32»

Київ – 2022 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання поширюється на розробку навчального програмно-апаратного комплексу для роботи з клавіатурою в STM32.

Область застосування – застосовується вищими навчальними закладами та Embedded компаніями, котрі проводять курси для глибокого розуміння ARM архітектури.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту по освітньо-професійної програми “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка навчального програмно-апаратного комплексу для роботи з клавіатурою в STM32.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література, описана мовою C та ARM архітектура для STM32 . Публікації в Інтернет з попередніх питань, документації та вивчення форумів та відеоматеріалів.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

- Розробка програмного коду для STM32 відповідно до ARM архітектури для плати STM32F4Discovery
- Автономність системи, її незалежність від іншого програмного забепечення

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows 8/10, Linux-дистрибутиви.
- STM32CubeIDE 1.7.0 та новіші версії
- STM32 ST-LINK Utility
- Virtual COM Port driver 1.5.0

5.3. Вимоги до апаратної частини

- Плата STM32F4Discovery або GL Embedded Starter Kit
- Процесор рівня Intel i3 і вище.
- Оперативна пам’ять не менше 4 ГБ.
- Вільне місце на жорсткому диску не менше 2 Gb.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2021
Створення та узгодження технічного завдання	15.01.2022
Вивчення літературних джерел	27.01.2022
Розробка технічного завдання	15.04.2022
Розробка програмної моделі	01.05.2022
Відлагодження програми та виправлення помилок	15.05.2022
Оформлення документації дипломного проєкту	01.06.2022

Пояснювальна записка
до дипломного проєкту
на тему: «Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32»

Київ – 2022 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

GPS (Global Positioning System) – дозволяє визначати положення та швидкість руху об'єкта на Землі.

POS (Point of Sale) – точка продажу.

IoT (Internet of Everything) – об’єднання інтернет речей у мережі.

ОС (Операційна система) – базовий комплекс програм.

DSP (Digital signal processing) – цифровий сигнальний процесор.

ESP32 – сімейство 32-бітних МК на кристалі.

STM32 – сімейство 32-бітних МК STMicroelectronics.

MCU (Micro Controller Unit) – мікрокомп’ютер на кристалі.

GPIO (General-purpose input/output) - інтерфейс для зв'язку між компонентами.

SPI (Serial Peripheral Interface) - послідовний периферійний інтерфейс, шина.

I2C - послідовна шина даних для зв'язку інтегральних схем.

I2S (Inter-IC Sound) – послідовна шина для цифрових аудіопристроїв.

UART (Universal asynchronous receiver/transmitter) - універсальний асинхронний приймач/передавач.

USART (Universal Synchronous-Asynchronous Receiver/Transmitter) - універсальний синхронний-асинхронний приймач/передавач.

IRQ (Interrupt ReQuest) – запит на переривання.

NVIC (Nested vectored interrupt controller) - модуль контролю переривань.

AHB (Advanced High Performance Bus) – удосконалена високопродуктивна шина.

APB (Advanced Peripheral Bus) – удосконалена периферійна шина.

HCLK – головний годинник процесора.

PCLK – периферійний годдинник процесора.

PLL (Phase-locked loop) – фазове автопідлаштування системи.

ARM (Advanced RISC Machine) - 32-бітна RISC архітектура процесорів.

CMSIS (Cortex Microcontroller Software Interface Standard) – набір інтерфейсних стандартів МК.

ADC (Analog-to-digital converter) - пристрій, що перетворює вхідний аналоговий сигнал в дискретний код.

ESP-IDF (Espressif IoT Development Framework) – фреймворк розробки IoT .

SNTP (Simple Network Time Protocol) – протокол синхронізації часу мережі.

OTA (Over-the-Air) – способи дистрибуції програмного забезпечення.

PWM (Pulse-width modulation) - Широтно-імпульсна модуляція.

USB (Universal Serial Bus) – стардарт роз’ємів і кабелів для передачі даних.

EEPROM (Electrically Erasable Programmable Read-Only Memory) - постійний запам'ятовувач, що програмується та очищується за допомогою електрики.

AVR - сімейство 8-бітових контролерів.

LDC (Low Duty Cycle) – переключання між 1 та 0 (циклічно).

HDMI (High Definition Multimedia Interface) – інтерфейс та кабель для передачі цифрових відео та аудіо.

eMMC (Embedded Multimedia Memory Card) – вид флеш пам’яті.

microSD (Secure Digital Memory Card) – вид флеш пам’яті.

FPU - модуль для роботи із числами з плаваючою комою.

CMOS (Complementary-symmetry/metal-oxide semiconductor) – невелика кількість пам'яті на материнській платі.

SRAM (static random access memory) – напівпровідникова оперативна пам'ять з довільним доступом.

PSRAM (Pseudostatic RAM) – динамічна RAM.

AES (Advanced Encryption Standard) – симетричний алгоритм блочного шифрування.

JTAG (Joint Test Action Group) – стандарт для перевірки плат.

SWD (Serial Wire Debugger) – відлагоджувач даних.

SWV (Serial Wire Viewer) – консоль відлагоджувача даних.

COM – послідовний порт комунікації.

IDR (Input Data Register) – вхідних регістр даних.

ODR (Output Data Register) – вихідних регістр даних.

EXTI (External Interrupt/Event Controller) – контролер зовнішніх переривань і подій.

USB OTG FS (full-speed) – повношвидкісний тип передачі даних.

USB OTG HS (high-speed) – високошвидкісний тип передачі даних.

GDB (GNU Debugger) – відлагоджувач Unix-систем.

DMA (Direct Memory Access) – взаємодія із прямим доступом до пам'яті.

HSE ((High speed external) – високочастотний генератор.

ВСТУП

У сучасному світі, де давно поширена глобалізація мережі інтернет – отримувати нові знання та навички на рахунок бажаної для вивчення теми, незалежно від сфери стає надзвичайно легко – незалежно чи то статті на тему програмування чи інших технологій.

Саме тому можливо знайти багато онлайн та офлайн платформ, компаній та менторів, котрі зможуть допомогти із вивченням програмування embedded систем, як приклад. Проте, не завжди всі варіанти є бездоганними. Наприклад, онлайн платформи зручні тим, що доступ до ресурсу можливий 24/7, проте коштує немалі кошти. Оффлайн курси – можуть не влаштовувати вас локацією проведення або часом проведення занять.Менторство є надзвичайно популярним на сьогодні, але часто підтримка порадами чи настановами від «старшого» не завжди приходить вчасно, оскільки даний вид допомоги надається певну обмежену кількість годин на тиждень.

Тобто, звідси можна зрозуміти, що проблематика у кожного із комплексів можлива і зачасту – неуникна. Тому, матиме сенс провести роботу не тільки щодо детального аналізу актуальностей та конкурентних продуктів –а й розробити власний проект, де з боку технічного забезпечення та обладнання для його здійснення – це мікроконтролери безпосередньо та/або периферія, котра необхідна для більш спеціалізованих ідей.

Як може здатися, для започатковування стартапу для розробки вбудованої системи потрібні великі матеріальні затрати, але це не завжди так. Більшість забезпечення можна придбати за безцінь, що не становить проблеми.

Проаналізувавши певні матеріали, порівнявши їхні характеристичні властивості, такі як переваги та недоліки – можна удосконалити розробку власного продукту.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

Спершу, аби здійснити аналіз та визначити найбільш ефективний і оптимальний варіант реалізації проекту із використанням вбудованої системи – варто розглянути загальне поняття вбудованих систем. І тільки опісля розгляду термінології можна зробити аналіз на рахунок актуальності даного проекту та розробки в цілому.

1.1 Поняття вбудованих систем

Embedded (вбудовані) системи – це система, що складається з апаратного забезпечення, прикладного програмного забезпечення та операційної системи реального часу. Це може бути невелика незалежна система або велика комбінована система. Використовуються системи для виконання спеціалізованих функцій, про це можна висновувати навіть судячи із назви. Тобто, як правило, застосовують їх не для широкого ряду функціональних завдань, як, наприклад, сучасні загальні системи, а для виконання конкретних задач на основі предметної області та проекту.

Оскільки, щороку тенденція використання вбудованих систем тільки зростає відносно попередніх років – було б доцільно розглядати збільшення дисциплін у вищих навчальних закладах, які мають пряме або дотичне відношення до вбудованих систем.

Актуальним завданням є застосування embedded систем не тільки в портативних користувацьких пристроях, на кшталт смарт-годинників, вимірювачів показників здоров’я (як-от наприклад, артеріального тиску), а й для складних та масштабних керуючих пристроїв для підприємств. Отже, капіталовкладення напряму залежать від складності – бо в залежності від архітектури мікропроцесорів та його параметрів, пам’яті та периферійних елементів можна сформувати різні цінові категорії. На початку вісімдесятих років двадцятого сторіччя потреба до збільшення кількості капіталовкладень в ринок embedded систем. Причиною були розвиток промислової та воєнної техніки, а також факт того, що інтелектуальна побутова техніка теж почала

займати нову нішу на економічному ринку. Тобто, великий запит на системи керування впливав на серйозний прогрес в технології виробництва напівпровідників, тощо.

Тобто, виходячи із цього, обчислювальні пристрої (тобто девайси із мікропроцесором або мікроконтролером), які не є комп'ютерами загального призначення, можна вважати вбудованою системою. Це можна трактувати так, що сучасний смартфон, як правило, не вважається вбудованою системою, але в ньому, швидше за все, є кілька вбудованих систем. До прикладу, електроніка, яка керує камерою та системою GPS, є вбудованою системою, оскільки вона виконує функцію в контексті більшої системи (смартфона).За аналогією, обчислювальний пристрій призначення (загального), такий як комп'ютер з Windows, може бути за потреби перетворений на вбудовану систему, якщо його призначення – суто виконання однієї функції. Приклади цього включають ПК, які призначені для запуску програм продажу (POS) або керування автоматизованими касовими автоматами (банкоматами).

Завдяки тому, що embedded система зазвичай повинна забезпечити функціонування певного ряду задач – відповідно з'являється можливість оптимізації програмного продукту та технічних характеристик. На основі статистичних та порівняльних графіків швидкодії програми при певних обставинах – у відповідності із місцями, де виникають «просідання» графіків – можна визначити, котрий із елементів потрібно удосконалити чи навіть замінити для подальшої оптимізації. Такий підхід до вирішення проблеми надає змогу збільшити ефективність вбудованої системи, і водночас витратити на це мінімальні затратні кошти, що є важливим, коли йде мова про подальше масове виробництво вихідного продукту.

Як правило, теперішні вбудовані системи базуються на основі мікроконтролерів. Мікроконтролером називається невелика за розміром інтегральна схема, яка надає доступ для управління операціями у вбудованій системі. Він складається із ядра процесора, пам'яті та периферії, яка може

надавати доступ для вводу-виводу даних, передачі даних до інших пристроїі тощо.

Як вже відомо, вбудовані системи стали революційними – тому, поки немає меж застосування їх в сучасній техніці. Тепер набувають популярності технології Інтернету речей – які здійснюють передачу інформації із датчиків (фізичний світ) та іншими вбудованими чи комп'ютерними системами через шифрований мережевий протокол. Більшість сфер використання вбудований систем – перегляньте на рисунку 1.1.

Також, для створення мереж, як правило, можуть бути використані вбудовані системи. У відповідності – існує два типи систем, котрі зможуть надавати наскрізні полуги – інфраструктурні (котрі, є базовими) та кінцеві системи. До переліку першої категорії належать комутатори, мости та роутери, здебільшого. В той час, до другої категорії можна віднести системи, котрі видимі для кінцевого користувача, такі як смартфони, модеми, тощо.

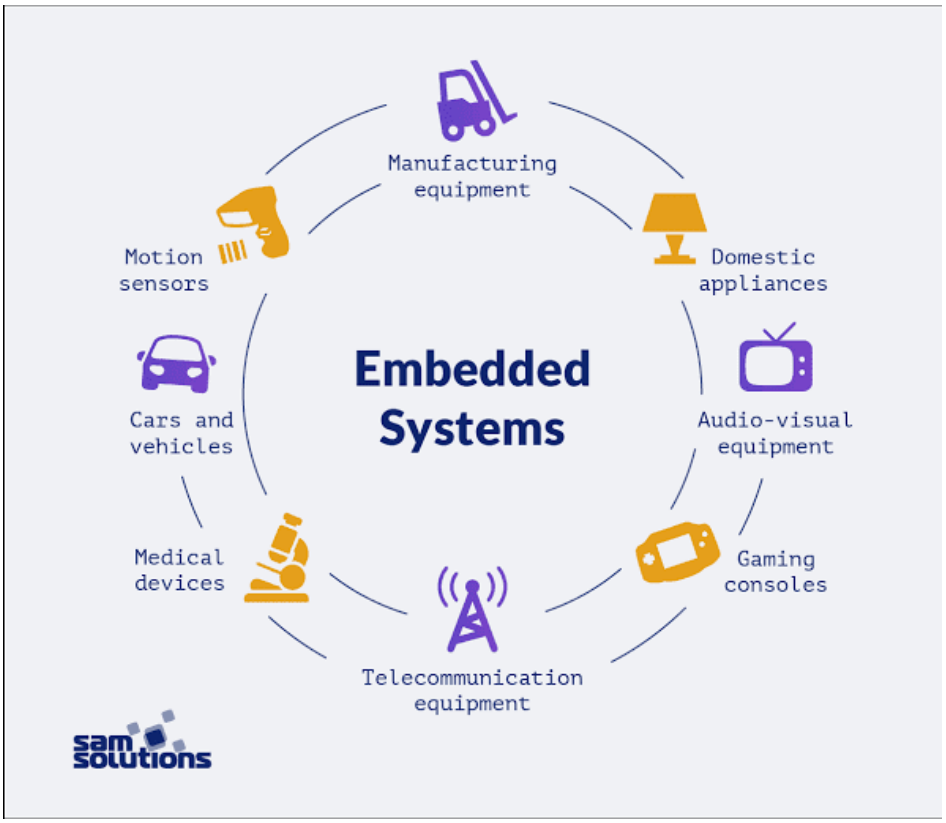


Рисунок 1.1 – Категорії розвитку вбудованих систем

Зазвичай, вбудовані системи використовуються в широкому ряді технологій у різних галузях промисловості, а саме:

- Автомобілі. Сучасні автомобілі, як правило, складаються з багатьох комп’ютерів або вбудованих систем, які надають змогу виконання різних завдань автомобілеві. Деякі з цих систем виконують деякі допоміжні функції, а інші забезпечують розважальні чи користувацькі функції. Деякі вбудовані системи автомобілей включають круїз-контроль, резервні датчики, контроль підвісок, навігаційні системи та системи подушок безпеки.
- Мобільні телефони. Вони складаються з багатьох вбудованих систем, включно із програмним та апаратним забезпеченням, інтерфейсом, операційною системою та засобами вводу/виводу даних (послідовна шина).
- Промислові машини. Як правило, містять вбудовані системи із використанням багатьох датчиків, але й самі можуть бути складними вбудованими системами. Часто на практиці використовуються системи автоматизації, які виконують функції моніторингу стану та контролю.
- Медичне обладнання. Вони можуть містити вбудовані системи, котрі складаються не тільки із датчиків але й обладнані механізмами керування. Медичне обладнання, як і промислові машини, повинні бути не тільки зручними для користувача, але й щоб помилки машин не загрожували здоров’ю людини, якщо їх можна запобігти. Це означає, що вони як правило мають складнішу ОС, відповідні тестові кейси та інтерфейси.
- Музична система. Реалізація музичної системи, яка зможе відтворювати більшість форматів мелодій - також є вбудованою системою (середній масштаб). Кодек (кодер і декодер) для відповідного формату має бути імплементований для кожного із форматів, що є досить таки складним завданням і вимагає алгоритмізації. Апаратне забезпечення повинно швидко реагувати на

зовнішні події, тому воно потребує високоякісних мікроконтролерів або DSP [1].

Важливість вбудованих систем константно зростає, оскільки зростає кількість сфер застосування, де вони використовуються. Впродовж довгого часу ці системи використовувались у багатьох важливих областях програм, таких як авіаційна техніка та системи управління дорожнім трафіком. Саме тому, їх використання ілюструє ціну працездатності вбудованих систем, особливо при розгляді наслідків виходу таких систем із ладу. До прикладу, вихід з ладу системи автоматичного пілота або можлива несправність гальмівної системи автомобіля ймовірно призведе до втрати життів; вихід з ладу електромережі може призвести як не до втрати життя – так до погіршення його якості, а вихід з ладу системи контролю виробництва заводу приведе до втрат доходу [1].

Отже, можна висновувати, що залежність від вбудованих систем вимагає впровадження кращих архітектурних та дизайнерських методів розробки, щоб задовольнити всі вимоги до продуктивності, та при цьому не забуваючи про надійність. Звідси, випливає причина появи потужної галузі, яка здійснює контроль цих процесів та розвиток, що невдовзі позначилося на економічному зростанні. А цей фактор - спонукав багатьох компаній світу до вирішення проблем, пов’язаних із впровадженням та розвитком систем. Проте, саме значний вклад у цій справі був від Європейської Комісії ARTEMIS. Програма розпочалася зі Стратегічного порядку денного досліджень і переросла до діяльності промислової асоціації ARTEMISIA.

1.2 Актуальність та доцільність розроблення

В умовах глобалізації мережі інтернет – немає обмежень щодо отримання нових знань та навичок щодо будь-якої із тем у будь-якій сфері, будь то наукові статті на тему біоінженерингу чи вбудованих технологій.

Наступний можливий проблематичний аспект має місце з боку огляду технічного забезпечення обладнання для здійснення проекту – це мікроконтролери, як мозок, безпосередньо, або периферія, котра необхідна для

більш спеціалізованих ідей. Проте, на відміну від того погляду, як може здатися, для започатковування стартапу із вбудованої системи не потрібні великі затрати на матеріальні ресурси. Більшість забезпечення можна придбати за декілька десятків у.о, що не становить проблеми. Тому, з боку забезпечення на технічному напрямку чи інформативному - для розвитку, у індивідуума немає особливих на те обмежень.

Проте, варто розглянути й інший, надзвичайно важливий пункт – наявність якісних матеріалів для підготовки – тобто, оффлайн курси чи відеокурси в Інтернет, наявність ментора для підготовки та хороша документація, де зібрані більшість описів схем та елементів продукту. Саме тому, питання виникатиме тільки тоді, коли буде нестача наведених пунктів.

На даний момент, є багато платформ із відеокурсами, котрі пояснюють роботу із платами – на ваш розсуд, якими саме – і надають посилання на документацію. Це, звісно, непоганий варіант, але він досить дороговартісний. Але не варто забувати про певні проблеми, із якими можна зіткнутися із даними відеокурсами.

Як правило, при виборі того чи іншого курсу – відбувається перегляд списку відеоматеріалів, їхніх тем, тощо. Проте, можна натрапити на курси із неякісним трактуванням та не зовсім тим вмістом, що необхідно. Адже автор не вказує які підтехнології будуть вказані в конкретному уроці. В такому випадку, варто звірятися із коментарями та популярністю такого виду проектів. Саме тому, розробка даного проекту може скласти конкуренцію на ринку.

Додатково, варто згадати про мову, якою вестиметься відеокурси, оскільки здебільшого немає можливості знайти відеоматеріали українською мовою. Єдині варіанти, котрі можливо занести на розгляд – це оффлайн курси, де необхідна локальна присутність або, як варіант, представлення матеріалів із детальним описом виконаних дій, де розроблена програма із схематичними матеріалами та додана документована інформація із описом необхідних параметрів для налаштування того чи іншого елементу.

Тепер, варто розглянути головних претендентів на конкурентноспроможність.

1.2.1 MCU1 Embedded C: Writing drivers for STM32 GPIO , I2C, SPI,USART from scratch

Даний курс описує досить таки багато функціональностей для написання драйверів на мікроконтролері STM32. Детальніше про сам опис курсу можна подивитися на рисунку 1.2.

Початкові вимоги для відеокурсу наступні.

- Базове розуміння мови програмування С
- Якщо немає попередніх знань із мікроконтролерами та відсутній досвід роботи із ними – порада завершити курс "Embedded C" для початківців

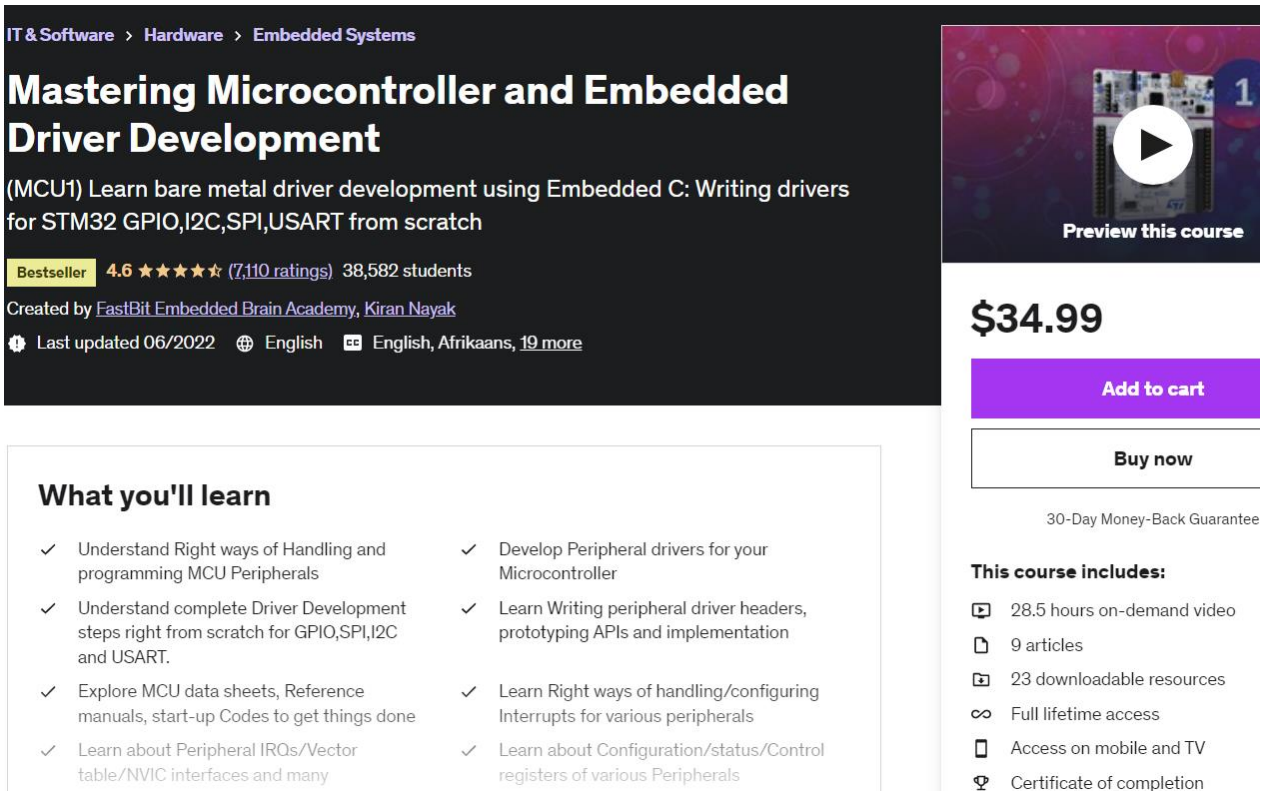


Рисунок 1.2 – Опис курсу Embedded C: Writing drivers for STM32

Від розробника наведений описовий перелік всіх тих технологій, котрі будуть використані у курсі. Пропонується такий перелік тем для пізнання.

- Дізнаєтеся про правильні способи обробки та програмування периферійних пристроїв СТМ

- Візьмете участь у безпосередній розробці периферійних драйверів для мікроконтролера
- Зрозумієте алгоритмізацію розробки драйверів для GPIO, SPI, I2C та USART
- Навчитеся писати заголовки периферійних драйверів, прототипувати API та реалізацію
- Ознайомитеся з таблицями даних, документацією, кодами запуску, щоб виконати завдання курсу
- Дізнаєтеся правильні способи обробки/налаштування переривань для периферійних пристроїв
- Дізнаєтеся про периферійні інтерфейси IRQ/Векторна таблиця/Інтерфейси NVIC та інші
- Докладніше отримаєте інформацію про регістри конфігурації стану/керування різних периферійних пристроїв
- Демістифікація за лаштунками робочих деталей SPI, I2C, GPIOs, USART, проведеться дослідження прихованих інтерфейсів шин MCU, годинників, їхніх конфігурацій, тощо.
- Зрозумієте способи ввімкнення / налаштування периферійних годинників, послідовних годинників та частоти передачі даних різних послідовних протоколів
- Знайдете детальну інформацію про протоколи шин MCUs AHB, APB
- Отримаєте знання про різні годинники MCU, такі як HCLK, PCLK, PLL, тощо
- Навчитеся захоплювати/декодувати/аналізувати сліди послідовних протоколів на аналізаторі Logic
- Дізнаєтеся про швидкі способи налагодження периферійних проблем із проведенням тематичних досліджень [2].

Також, не слід забувати про певні недоліки, із котрими доведеться зіткнутися.

- Мова ведення курсу – англійська

- Вартість курсу
- Необхідність купити додаткове устаткування для відлагодження системи

1.2.2 With hands on coding C Programming and assembly on ARM Cortex M Processor based Microcontroller

Даний курс описує досить загальну структуру програмування на ARM процесорі типу M3/M4, котрі часто використовуються компанією STM32. Для устаткування мікроконтролерів. Детальніше опис курсу наведено на рисунку 1.3.

Початкові вимоги для відеокурсу наступні.

- Базове розуміння мови програмування С
- Якщо немає попередніх знань із мікроконтролерами та відсутній досвід роботи із ними – порада завершити курс "Embedded C" для початківців

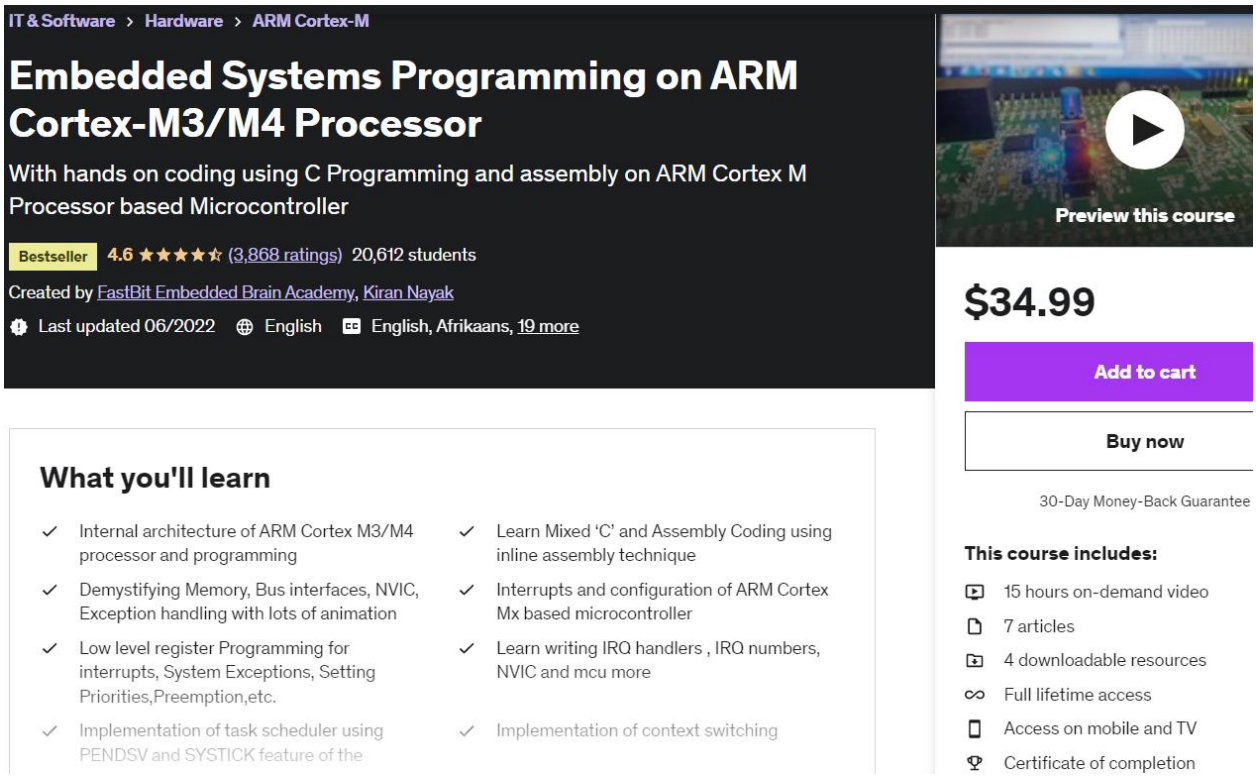


Рисунок 1.3 – Опис курсу Embedded C ARM Cortex M Processor

Розробником наданий список технологій, використаних у авторському курсі.

- Внутрішня архітектура процесора ARM Cortex M3/M4 і його програмування
- Вивчання змішаної мови C із кодуванням зборки за допомогою вбудованого асемблювання
- Демістифікація щодо пам'яті, інтерфейсів шин, NVIC, обробка винятків з великою кількістю анімаційних слайдів
- Переривання як поняття та конфігурація мікроконтролера на основі ARM Cortex Mx, де x – 3 та 4
- Низький рівень реєстрації програмування для переривань, системних винятків, встановлення пріоритетів, preemption і т.д.
- Розробляння обробників IRQ, NVIC для MCU
- Реалізація планувальника задач із використанням функції PENDSV і SYSTICK у процесора
- Здійснення перемикання контексту
- Вивчення та запис скрипт linker та файлу запуску MCU from scratch
- Bare-metal вбудований процес побудови архітектурного рішення системи
- Винятки несправностей процесора та імплементація обробника несправностей із аналізом несправностей
- Стандарт стеку та AAPCS
- Вбудовану збірка, функції та атрибути змінних та розділ компіляції із gcc інструментом [3].

Також, варто згадати про недоліки, із котрими доведеться зіткнутися, як і в попередньому підпункті.

- Вартість курсу
- Мова ведення– англійська
- Підійде тільки для початківців в сфері embedded розробки

1.2.3 ARM Cortex-M Bare-Metal Programming

Якщо необхідно зрозуміти структуру програмування на ARM Cortex M процесорах, є змога скористатися даним курсом, проте не варто забувати про

певні недоліки, які перевершують отримані переваги від відеокурсу. Але, все ж варто роздивитися курс як альтернативу рішенню - опис курсу зображено на рисунку 1.4.

Деякі початкові вимоги для перегляду курсу наведені нижче.

- Використання Keil uVision 5 IDE та набір безкоштовних інструментів
- Курс не передбачає попередніх знань про розвиток Cortex-M
- Стартовий майданчик TIVA C - TM4C123 Board

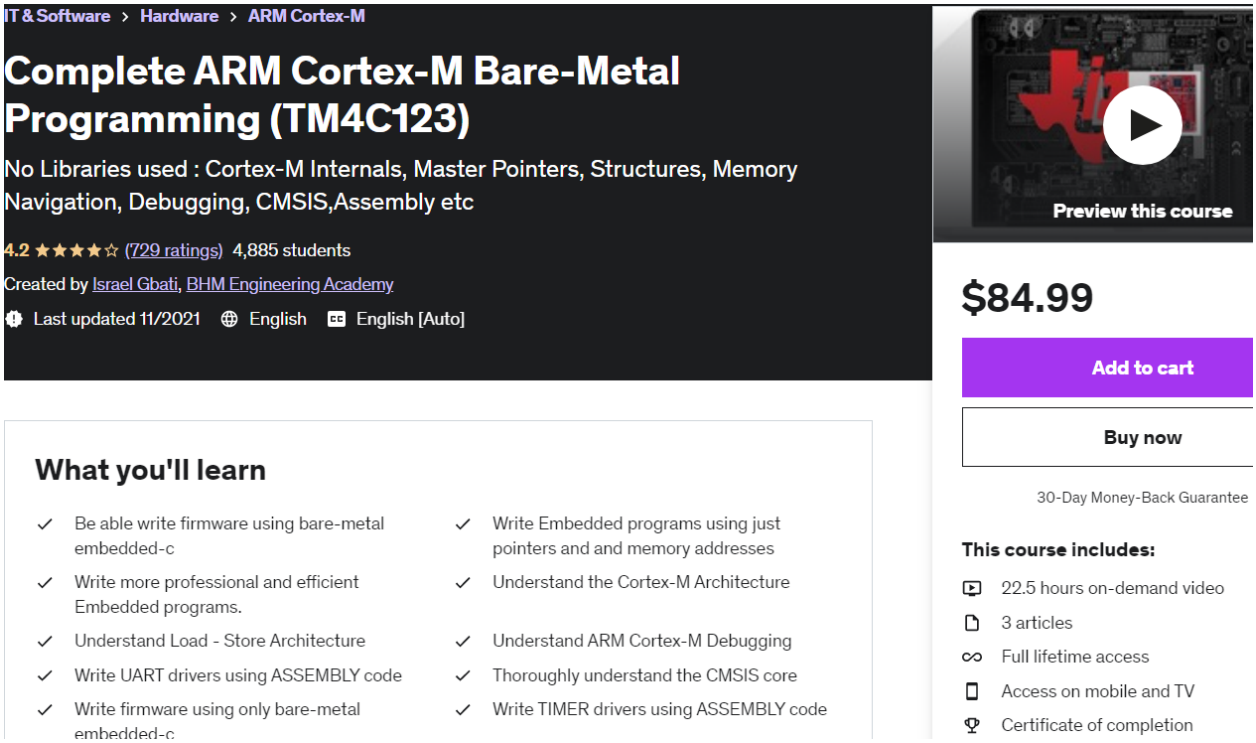


Рисунок 1.4 – Опис курсу ARM Cortex M програмування

Основні пункти, котрі розглянуті у курсі, будуть представлені автором курсу нижче.

- Можливість запису прошивки за допомогою embedded-c bare-metal
- Запис embedded програм, використовуючи лише вказівники-покажчики та адреси пам'яті
- Імплементація більш професійних та ефективних вбудованих програм
- Детальний розгляд архітектури Cortex-M «під капотом»
- Зрозумієте Load-Store архітектуру
- Повне розуміння того, як влаштоване ядро CMSIS

- Зрозумієте ARM Cortex-M налагодження
- Запис драйверів UART за допомогою ASSEMBLY коду
- Запис прошивки за допомогою тільки embedded bare-metal
- Запис драйверів TIMER за допомогою ASSEMBLY коду
- Записати переривання драйверів за допомогою embedded bare-metal
- Запис драйверів аналогового до цифрового перетворювача (ADC) за допомогою embedded bare-metal [4].

Але на противагу всім вищезазначеним перевагам, не варто забувати про мінуси системи курсу. Окрім надзвичайно високої вартості, до списку додається англомовна система.

1.2.4 IoT розробка додатків із ESP32

В даному курсі розповідається про створення IoT додатків на платі ESP32 із використанням ESP-IDF – офіційним фреймворком для ESP32, котрі забезпечує самодостатній SDK для загальної розробки додатків на цій платформі, програмуючи мовою C чи C++.

Працюючи над цим проектом, програмуючи крок за кроком на уроках – буде змога розширювати додаток, розширювати WLAN програму та інтегрувати хмарний фреймворк за допомогою ESP-IDF або аналогічної до неї. Використовувати при цьому буде зручно документацію Espressif, пошук посилань на API та відповідних функції, тощо, що може знадобитися для проекту.

Крім того, в цьому курсі не буде тільки зосередження на теорії, оскільки це практичний курс прикладного програмування, де важливо навчитися пишучи код. Окрім цього, буде коротко представлено довідкову інформацію щодо інтерфейсу прикладного програмування ESP-IDF, короткий огляд вимог до програмного коду для кожного етапу уроків. Вони будуть описувати, що буде досягнуто і яким чином буде досягнено цілей за допомогою фреймворку. Детальніше переглянути веб-сторінку можна на рисунку 1.5.

Перед тим, як приступити до перегляду відеоматеріалів необхідно наступне.

- Комплект для розробки ESP32
- Датчик DHT22, при бажанні отримати дані про температуру та вологість
- RGB LED, для відображення кольорів стану світлодіода
- Дроти-перемички і bread board
- Знання мови програмування C

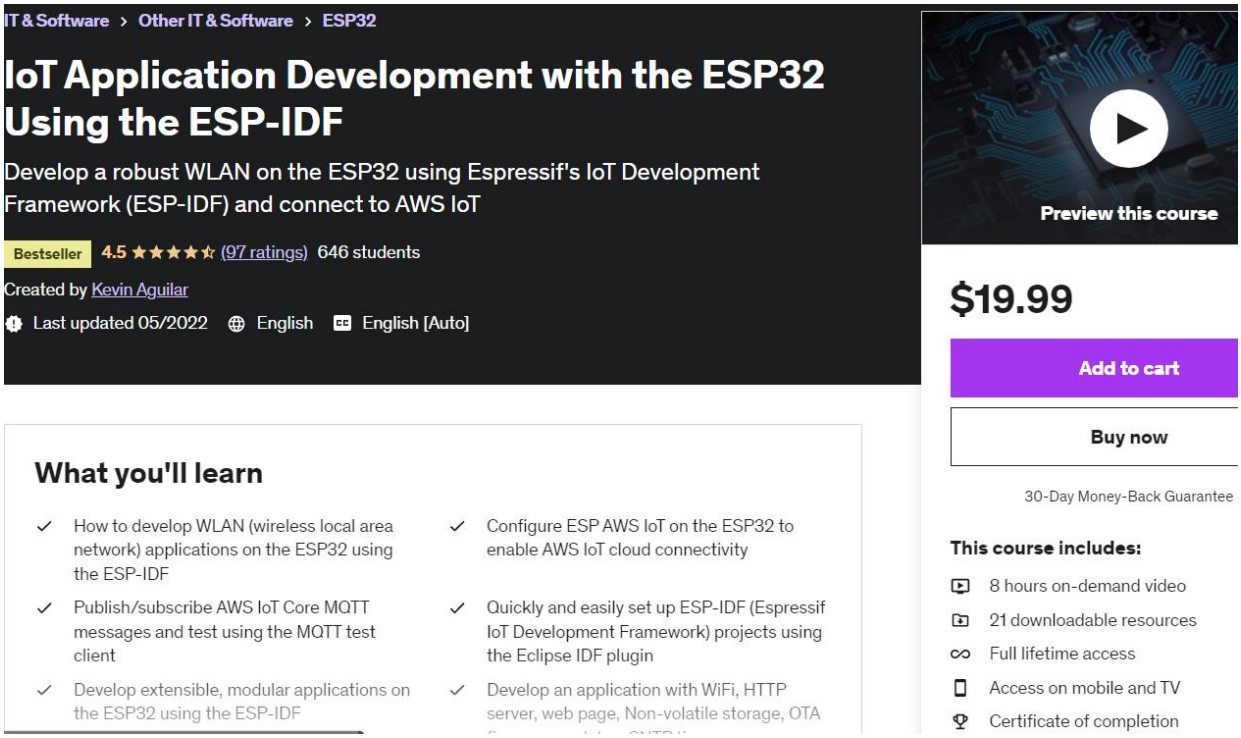


Рисунок 1.5 – Опис курсу ESP32 IoT програмування

Головні пункти, які курс детально обговорюватиме міститимуть наведені теми.

- Як розробити програми із WLAN (бездротова локальна мережа) на ESP32 через ESP-IDF
- Налаштуйте AWS IoT на ESP32, щоб увімкнути хмарне з'єднання
- Публікуйте/підписуйте повідомлення AWS IoT Core MQTT та тестуйте за допомогою тестового клієнта
- Швидко та легко конфігуруйте проекти ESP-IDF (Espressif IoT Development Framework) за допомогою плагіна Eclipse IDF
- Розробка розширюваних модульних додатків на ESP32 з використанням ESP-IDF

- Розробіть додаток з WiFi, HTTP-сервером чи сторінками веб додатку, енергонезалежним сховищем, оновленнями прошивки OTA, синхронізацією часу SNTP, RGB LED чи кнопкою з перериванням
- Використовуйте FreeRTOS для керування завданнями та взаємодією між завданнями
- Способи побудови, прошивки і контролю додатку в рамках IDF версії Eclipse
- Метод розробки веб-сторінку для відображення даних, підключення ESP32 до точки доступу, відключення ESP32, завантаження нової прошивки (оновлення OTA)
- Ви дізнаєтеся, як використовувати примітиви FreeRTOS - черги повідомлень, групи подій і семафори
- До кінця курсу ви попрактикуєтеся в розробці розширюваного додатка WLAN через ESP-IDF [5].

На диво, вартість курсу не є надто високою, з недоліків тільки те, що курс ведеться англійською (якщо це можна вважати недоліком).

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було проаналізовано термінологічне поняття вбудованих систем, їхню роль у розвитку технологій по цілому світі та сфери, для яких відсутність embedded технологій призвела б до величесних проблем.

Також було проведено роботу щодо аналізу актуальності та доцільності розробки даного проекту як навчального комплексу роботи із STM32, та проагалізовано ситуацію на ринку сучасних відеоматеріалів, курсів та детально розглянуто та описано кожен вибраний курс по актуальній темі.

На основі розробленого дослідження було визначено, що матеріалів українською практично немає, більшість матеріалів, котрі підходять за стеком технологій та відповідною платою наявні тільки англійською мовою, і до того ж коштують чимало. Саме тому можна зробити висновок на рахунок актуальності – реалізація однозначно матиме сенс.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМИ УПРАВЛІННЯ

2.1 Використані технології, обгрунтування вибору технологій

Для обробки вхідної та вихідної інформації потрібен, як мінімум, вбудований пристрій, який є оптимальним варіантом для реалізації. Вибір апаратного засобу вплине на швидкість обробки даних, а також, якість отриманих чи відісланих даних по інтерфейсах. Тому, варто розглянути деякі ринкові пропозиції, які найбільш часто використовують для схожих цілей.

2.1.1 Arduino

Напевне, немає розробника, який би не чув про дану плату. Вона є найпопулярнішою із всіх, оскільки є доволі простою при програмуванні та абсолютно недорогою, в порівнянні із іншими конкурентами та має open-source платформу. Зазвичай, використовується для отримання даних із конкретних датчиків та роботи із деякими елементами (напівпровідники).

Як було згадано вище, програмування плати є простим, навіть для початківців. Команда розробників плати Arduino створили для неї свою мову програмування та середовище розробки програмного забезпечення - ArduinoIDE. Більше того, Arduino, як мова програмування, є фреймворком, надбудованим над C++ із використанням 2 основних функцій – setup() та loop(). До того, програмування можна назвати гручким, оскільки запуск написаних програм можна реалізовувати на різних операційних системах – Windows, Linux та MacOS [6].

Як правило, пристрій чи система, яка виконана на основі Arduino – складається із базової плати із мікроконтролером і доданого до неї, модуля розширення, котрий називається shield. Також, для безпосереднього програування знадобиться кабель USB, блок живлення і ПК.

Існує декілька рішень на основі вибору різних мікроконтролерів, які представлено таблицею 2.1.

Таблиця 2.1 – Порівняльні характеристика МК Arduino

Тип плати Arduino	**Тип процесора**	**Напруга/ живлен-ня, [В]**	**Тактова частота CPU [МГц]**	**Входи/ анало-гові виходи**	**Цифрові IO/ к-сть PWM**
LilyPad USB	Atmega 32U4	3,3/3,8…5	8	4/0	9/4
Mega 2560	Atmega 2560	5/7…12	16	16/0	54/15
Micro	Atmega 32U4	5/7…12	16	12/0	20/7
Uno	Atmega 328P	5/7…12	16	6/0	14/6
Zero	ATSAMD 21G18	3,3/7…12	48	6/1	14/10
Nano	Atmega 168; Atmega 328P	5/7…9	16	8/0	14/6
Nano 33 IoT	SAMD21 (Cortex-M0+)	3,3/3,6…21	48	8/1	14
Portenta H7	STM32H747 (Cortex-M7+M4); Murata 1DX WiFi, Bluetooth 5.1	3,3/5	480	7/2	15/8

Кінець таблиці 2.1

Тип плати Arduino	**SRAM [кБ]**	**Flash [кБ]**	**USB**	**UART**	**EEPROM [кБ]**
LilyPad USB	2,5	32	Micro	-	1
Mega 2560	8	256	Тип B	4	4
Micro	2,5	32	Micro	1	1
Uno	2	32	Тип B	1	1
Zero	32	256	2×Micro	2	-
Nano	1; 2	16; 32	Mini	1	0.512; 1
Nano 33 IoT	-	32	256	Да	-
Portenta H7	1MB	2MB	Да	4	-

Детальніше поглянути на мікроконтролери Arduino – можна на рисунках 2.1 та 2.2, де зображені Arduino UNO та Nano відповідно. Переглянувши характеристики, можна навіть побачити неозброєним оком – котра із них виглядає більш потужною.

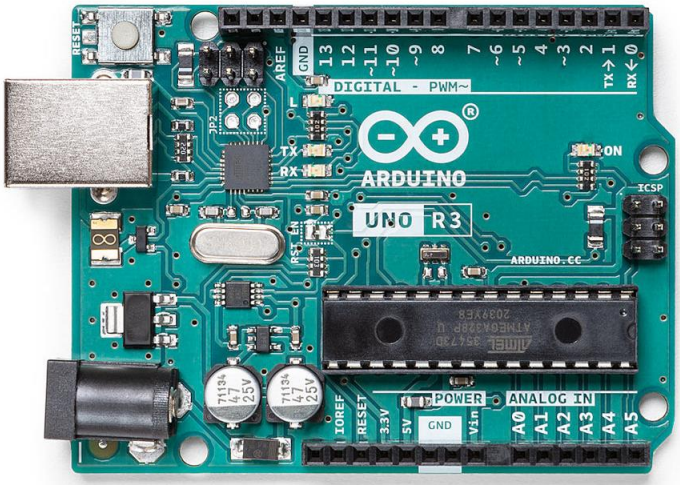


Рисунок 2.1 – Мікроконтролер Arduino Uno

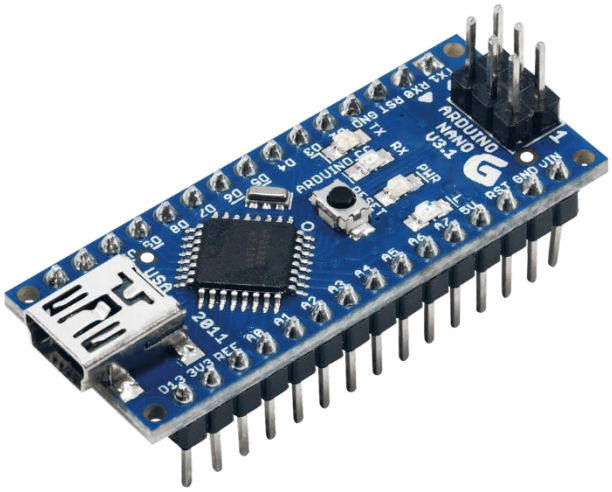


Рисунок 2.2 – Мікроконтролер Arduino Nano

Залежно від типу - плати оснащені роз'ємами - гнізда під goldpin або точки пайки, які можуть використовуватися не тільки для підключення плати, але також для її кріплення на друкованій платі embedded, якщо мікрокомп'ютер Arduino виконує роль головної одиниці. Кожна з плат має збережений у пам'яті мікроконтролера bootloader, який використовується для програмування процесора in-circuit (без випайки зі схеми) за допомогою простого вибору параметрів з меню середовища ArduinoIDE.

Актуальний список базових плат Arduino наведено у таблиці вище. Більшість використовують мікроконтролери з ядром AVR, але також серед них можна знайти процесор марки Intel і SAM21, оснащений ядром ARM Cortex - M0+. При цьому варто зазначити, що в таблиці не представлені плати, оснащені процесорами Espressif Systems (наприклад, популярним ESP8266), хоча вони також можуть бути запрограмовані за допомогою ArduinoIDE.

При виборі плати додатка необхідно керуватися можливостями встановленого на ній мікроконтролера. Окремі одиниці відрізняються розмірами доступної пам'яті, швидкістю роботи ядра та оснащенням функціональними блоками, такими як інтерфейси, таймери, генератори PWM тощо. Також варто звернути увагу на виходи на платі, оскільки деякі з них не мають роз'ємів, але призначені для припаювання. Проте, зазвичай, не буде необхідності припаювати елементи паяльником. Більшість елементів,

необхідних для сучасних проектів можна зібрати на макетній дошці (модулі розширення), просто приєднавши виводи до неї або залучити перемички.

Як початкову точку для початківців часто рекомендують Arduino Uno. Він оснащений USB-роз'ємом, за допомогою якого можна під'єднати плату до ПК і пересилати програмне забезпечення одним клацанням миші. Мікроконтролер ATmega328, встановлений на платі, має достатній обсяг пам'яті, а також необхідне апаратне обладнання для реалізації програм для контролю та управління.

Тактова частота ядра - 16 МГц, що дає машинний цикл тривалістю 62,5 нс, а ядро AVR, яке тут наявне, виконує більшість команд в одному машинному циклі. У міру набуття навичок та досвіду можна вибирати такі варіанти, такі як Arduino Due, Mega 2560 та інші. Опісля, варто звернути увагу на модель Arduino Nano, яка є мініатюрною версією великих схем, але де відсутні стабілізатор напруги і повнорозмірний USB-порт [6].

Arduino Nano, однак, був оснащений тим же 8-бітним процесором, що і плата Uno, при вражаючій мініатюризації. Для серії Nano розмір PCB складає 18x45 мм. Важливо відзначити, що, незважаючи на апаратні зміни, можна використовувати все те ж програмне середовище ArduinoIDE.

Отже, головними перевагами при виборі Arduino:

- доступна ціна, нижча при порівнянні з аналогами;
- cross-platform (програма працює на Windows, Mac OSX, Linux);
- нескладність створення програм;
- програмне забезпечення з відкритим кодом (open-source);
- розширення функціональності при потребі;

2.1.2 ESP32

З розвитком технологій з’явилися інноваційні ідеї та відбулося впровадження нових проектів, тому набула поширення концепція – Інтернету речей або IoT. Це платформа підключень, де кілька «речей» чи пристроїв підключені через Інтернет для обміну необхідною інформацією.

У спільноті проекти IoT, як правило, зосереджені на додатках для автоматизації та створення розумного дому, проте комерційні і промислові проекти IoT складаються із набагато складніших реалізації, як-от наприклад машинне навчання, штучний інтелект або бездротові несенсорні чи сенсорні мережі. Важливим елементом у цьому не залежно від виду та складності проекту є той факт, що проект IoT неможливий без підключення до Інтернету. Ось де мають перевагу такі плати, як ESP8266 і ESP32.

Отже, із сказаного вище, для створення мобільних пристроїв, смарт електроніки та додатків Інтернету речей (IoT) виникла потреба у невеликих за розміром схемах невеликої потужності та бажано, за доволі прийнятну ціну.

Цим скористалася китайська компанія Expressif Systems, яка створила серію мікроконтролерів ESP32, котрі є системами на кристалі (SoC) із додатково інтегрованими контролерами Wifi та Bluetooth (двох режимів).

Ця система включає в себе наявність всіх сучасних характеристик мікросхем малої потужності, включно із дрібнозернистим підсилювачем тактової частоти, декілька режимів живлення плати та динамічне масштабування потужності.

Для прикладу, щоб інтегрувати низьке споживання енергії – було додано IoT сенсор хаб, який «прокидається» із певною частотою тільки за певної умови – bool (true). Тобто, Low-duty Cycle (LDC) використано для того, аби мінімізувати кількість енергії, яку витрачає чіп. Також, варто додати, що й вивід підсилювача потужності також можна регулювати. Завдяки цій особливості виникає компромісне рішення між дальністю зв’язку, швидкістю передачі даних та споживанням енергії.

Ще одна цікава річ, яку варто знати про ESP32 — це те, що він виготовлений за допомогою 40-нм стандартом TSMC за надмалопотужною технологією. Звідси випливає, що розробка додатків, котрі працюють від батарейок, таких як переносні пристрої, аудіотехніка, радіоняні, розумні пристрої, годинники і т.п., за допомогою ESP32 має бути дуже простим.

Тепер, необхідно провести аналіз характеристик плат ESP, котрий наведений у таблиці 2.2, а рисунок кожної із плат зображений на рисунку 2.3.

Таблиця 2.2 - Порівняльні характеристики плат ESPxxxx [7]

	ESP32	ESP8266
MCU	Xtensa Dual-core LX6 with 600 DMIPS	Xtensa Single-core L106
Кількість ядер	2	1
Архітектура	32 bit	32 bit
Частота CPU	160 MHz	80 MHz
WIFI 802.11 b/g/n	HT40	HT20
Bluetooth	+	-
RAM	512 KB	160 KB
FLASH	16 MB	16 MB
GPIO Pins	36	17
ADC Pins	18	1
DAC Pins	2	0
HW / SW PWM	None/16 channels	None / 8 channels
SPI	4	2
I2C	2	1
UART	2	2
I2S	2	2
CAN	+	-
Ethernet MAC Interface	+	-
Touch Sensor	+	-
Temperature Sensor	+	-

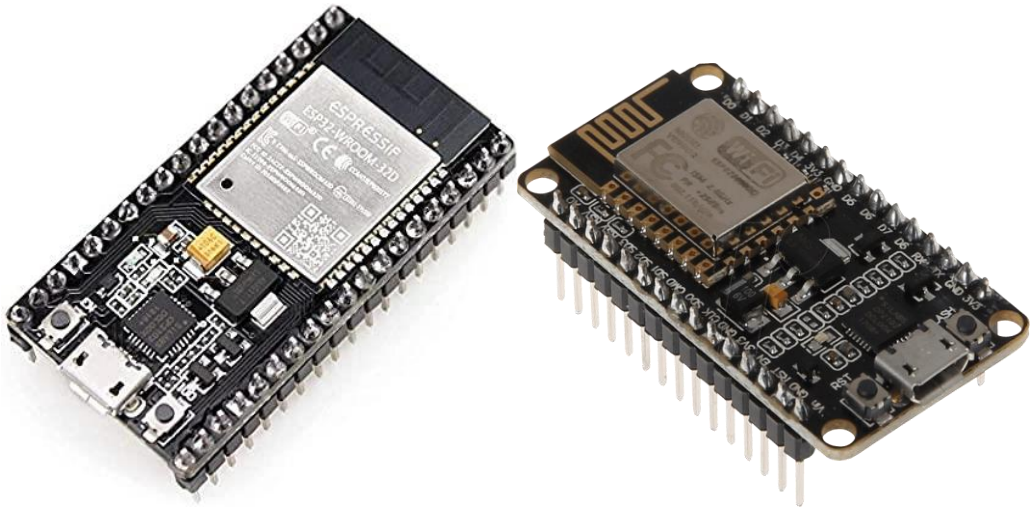


Рисунок 2.3 – Мікроконтролери ESP32 та ESP8266

Якщо вам необхідно додати Wi-Fi з'єднання у своєму проекті і там не задіяна складна логіка - то ESP8266 є чудовим варіантом. Але якщо ви хочете створити повну систему з підключенням Wi-Fi, Bluetooth, АЦП високої роздільної здатності, ЦАП, Serial підключення та багатьма іншими функціями, то ESP32 — найкращий вибір. Опис пінів та їхнє можливе призначення наведено на рисунку 2.4.

Тепер можна розглянути різні способи програмування даної плати. ESP32 буде більш зручним для користувача, якщо його можна запрограмувати декількома способами. Тому не дивно, що ESP підтримує декілька популярних середовищ програмування.

Одні з найбільш часто використовуваних середовищ є:

- Arduino IDE
- VS Code / PlatformIO IDE
- LUA
- MicroPython
- JavaScript
- Espressif IDF (IoT Development Framework)

Як правило, Arduino IDE вже знайоме багатьом програмістам, адже більшість із них починали програмувати в цьому середовищі на Arduino платі

– тому, дане IDE знаходиться у топі використання для ESP32 у проектах. Але точно можна спробувати й інші, якщо вам знадобляться плагіни (VS Code).

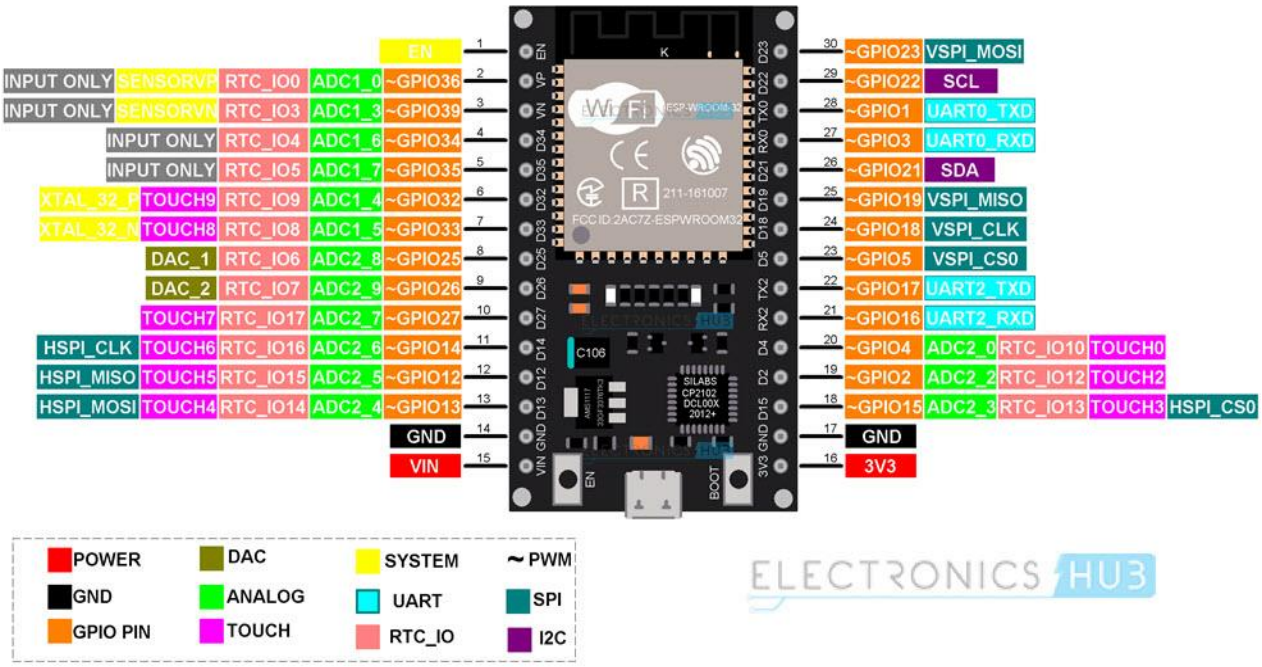


Рисунок 2.4 – Опис пінів GPIO на платах ESP32

Тепер розглянемо переваги та недоліки плат у порівнянні.

Перш за все, ціна. ESP8266 дешевше, ніж ESP32. Хоча він не має такої кількості функціональних можливостей, він чудово підійде для більшості простих проектів IoT DIY. Проте плата має деякі обмеження у відображенні GPIO, їй просто може не вистачити пінів.

Наступним критерієм є потужність. ESP32 набагато потужніший, ніж ESP8266, окрім вищезгаданих більших кількостях GPIO з декількома функціями, швидшим Wi-Fi і підтримки Bluetooth.

Існує помилкове твердження, що з ESP32 важче працювати, ніж з ESP8266, бо плата складніша. Навпаки, як показує практика, ESP32 так само легко запрограмувати, як і ESP8266, особливо якщо ви маєте намір програмувати його за допомогою Arduino мови або MicroPython.

Також варто зазначити, що ESP8266 «старіший» за ESP32, деякі бібліотеки та функції були розроблені для ESP8266 початково, тому більше ресурсів буде для роботи із ESP8266 (форуми, де люди із такими ж проблемами знаходять способи вирішення, тощо). Проте з часом ESP32

набуває популярності - тому ці відмінності з точки зору розробки та бібліотек анігілюються найшвидшим часом.

2.1.3 Raspberry PI

Raspberry Pi є серією одноплатних комп'ютерів, виготовлених Raspberry Pi Foundation, благодійною організацією Великобританії, яка має на меті навчати людей комп'ютерній сфері та полегшувати доступ до комп'ютерної освіти. Отже, це недорогий комп'ютер розміром із кредитну картку (85 x 55 mm), який підключається до монітора комп'ютера чи телевізора (вивід інформації) та для вводу використовує стандартну клавіатуру та мишку. Незважаючи на невеликий розмір, це потужний пристрій, який дає змогу досліджувати комп'ютери та навчитися програмувати, починаючи мовою Scratch та закінчуючи мовою Python. Він може робити все, що очікувано від настільного комп'ютера - від перегляду веб-сторінок в Інтернеті та відтворення відео доволі високої роздільної здатності до створення електронних таблиць, обробки тексту та «неважких» ігор.

Окрім того, Raspberry Pi надає можливість взаємодіяти із зовнішнім світом і використовується в низці проектів цифрових виробників, від музичних пристроїв і батьківських детекторів для моніторингу до метеостанцій та детекторів із інфрачервоною камерою із розпізнаванням облич.

Raspberry Pi був запущений у виробництво в 2012 році, і відтоді було випущено декілька ітерацій – варіацій пристроїв. Оригінальний Pi мав одноядерний процесор з частотою 700 МГц та оперував 256 МБ оперативної пам'яті. Для порівняння – тепер остання модель має чотириядерний процесор з тактовою частотою понад 1,5 ГГц і оперує 4 ГБ оперативної пам'яті. Ціна Raspberry Pi коливалася в межах близько 35 доларів США, зокрема Pi Zero коштував тільки 5 доларів. Приклад плати Raspberry Pi 3 можн апобачити, роздивившися рисунок 2.5.

Розробники використовують Raspberry Pi у своїх проектах не тільки для створення апаратних проектів по автоматизації дому, але й для впровадження

кластерів Kubernetes і Edge обчислень – ба навіть використання їх у промислових додатках.

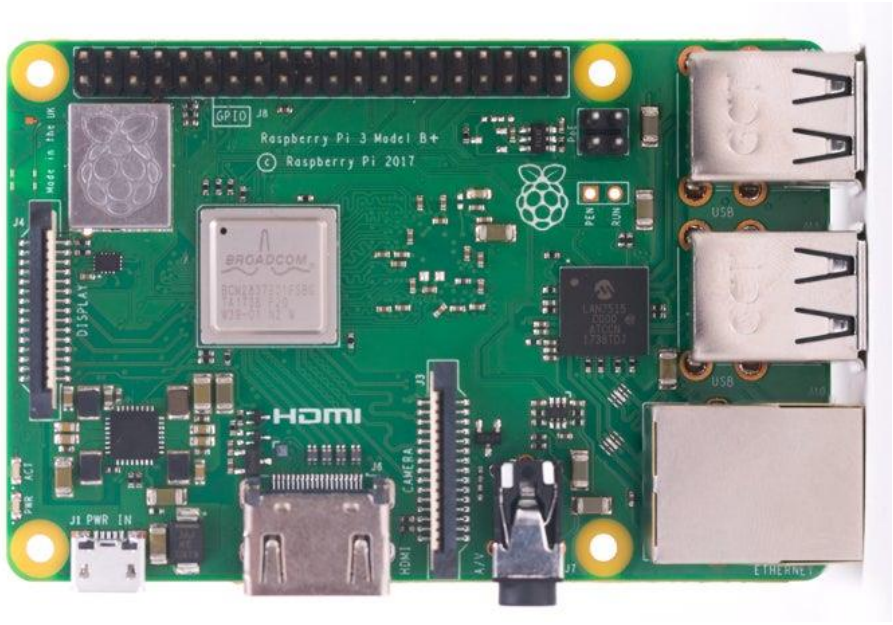


Рисунок 2.5 – Комп’ютер Raspberry Pi

Raspberry Pi — це відносно дешевий комп’ютер, який працює на операційній системі Linux, забезпечує набором пінів GPIO (загального призначення введення/виведення), які дозволяють керувати електронними компонентами для обчислень та досліджувати Інтернет речей (IoT) [8].

У лінійці Raspberry Pi було багато поколінь: від Pi 1 до 4 і навіть Pi 400, також були моделі A та B для більшості поколінь. Модель A була дешевшим варіантом і, як правило, мала меншу кількість оперативної пам’яті та портів (наприклад, USB та Ethernet). Pi Zero є додатковим продуктом оригінального покоління (Pi 1), зробленим ще меншим і дешевшим.

Raspberry Pi працює в екосистемі з відкритим вихідним кодом: в основі працює Linux (різноманітні дистрибутиви), і його основна операційна система, Pi OS, є open-source продуктом. Raspberry Pi Foundation робить внесок до розробки Linux ядра та працює над різними проекти з відкритим вихідним кодом, а також випускає більшу частину власного програмного забезпечення з відкритим кодом.Схеми Raspberry Pi регулярно випускаються у додатку до документації, проте плата не є відкритим апаратним забезпеченням.

Тепер варто розглянути характеристики кожної із моделей у таблиці 2.3.

Таблиця 2.3 – Порівняльні характеристика RaspberryPi [8]

	RPi3 Mod B+	**RPi3 ModA+**	**RPi Zero**	**RPi3 Compute**
SoC Type	Broadcom BCM2837B O	Broadcom BCM2837B O	Broadcom BCM2835	Broadcom BCM2837
Core Architecture	Arv8-A (64/32 bit)	ARMv8-A (64/32bit)	ARMv6Z (32 bit)	ARMv8-A (64/32 bit)
N Cores	4	4	1	4
GPU	Broadcom VideoCore IV 1080p60	Broadcom VideoCore IV 1080p60	Broadcom VideoCore IV 1080p30	Broadcom VideoCore IV 1080p60
CPU Clock	1.4 GHz QuadCore ARM Cortex-A53	1.4 GHz QuadCore ARM Cortex-A53	1 GHz Single Core ARM1176JZ	1.2GHz QuadCore ARM Cortex-A53
RAM	1 Gb	512 Mb	512 Mb	1 GB
ROM	MicroSD	MicroSD	MicroSD	MicroSD
USB ports	4 x USB	1 x USB	1 x Micro USB (OTG)	1 x USB
Ethernet	10/100/1000 Mbit/s Ethernet Port	-	-	-
WiFi	OnBoard WIFI 802.11a Dual	OnBoard WIFI 802.11a Dual	-	-

Кінець таблиці 2.3

Bluetooth	OnBoard Bluetooth 4.2/4.1/2.0 LS	OnBoard Bluetooth 4.2	-	-
Video Output	HDMI 3.5 Composite DSI	HDMI 3.5 Composite DSI	Mini HDMI Composite via PCB	HDMI 2xDSI Composite
Audio Output	I2S HDMI 3.5 Composite	I2S HDMI 3.5 Composite	I2S Mini HDMI	I2S HDMI Composite/Analog
Camera Input	15 Pin CSI	15 Pin CSI	15 Pin CSI	2x15 Pin CSI
GPIO Pins	40	40	40	40

Попри велику потужність обчислювальної системи, оскільки тут наявний RISC процесор на 4 ядра, тут наявне використання енергоефективних технологій.

Отже, підведемо підсумок щодо плюсів та мінусів Raspberry Pi комп’ютера.

Основні переваги:

- Широка підтримка периферії
- Можинні сенсори завдяки великій кількості GPIO пінів
- Підтримка більшості мов коду – C/C++/C#, Java Python, Ruby
- Швидкий процесор
- Використання як портативного комп’ютеру

Декілька недоліків:

- Відсутність eMMC внутрішнього сховища – використання microSD (менша швидкодія пам’яті)
- Відсутність графічного процесору

- Перегрівання при довгому використанні плати
- Швидкий процесор
- Неможливо використовувати Windows OS

2.1.4 STM32F4-DISCOVERY

Мікроконтролери STM32 на думку багатьох розробників та науковців вважаються одним із найкращих рішень для вбудованих систем та систем обробки сигналів (реальний режим роботи). Головною перевагою є вбудований цифровий сигнальний процесор ARM Cortex – який створений для вирішення задач сигнальної обробки, наприклад фільтрування, пошуку та перетворень Фур'є.

Серія ядер ARM Cortex для STM32 – M0, M0 +, M3, M4 і M7 розроблена для вбудованих програм, кожна із яких вимагає високої продуктивності, низької вартості та збалансованого енергоспоживання. 32-розрядний мікроконтролер серії STM32 базований на процесорі Cortex-M3 [9].

Він може підтримувати багатоспектрові 32-розрядні додатки, включаючи ефективні та високопродуктивні функції в режимі реального часу, цифрову обробку сигналів і роботу на низькій напрузі. У той же час процесор має повністю інтегровану та доволі просту у використанні розробку. На основі платформи STM до відповідності вимогам керування у реальному часі є п'ть варіантів програмування: μC/OS-II, μClinux, eCos, FreeRTOS і Dujiangyan OC (djyos).

μC/OS-II є пріоритетною багатозадачною ОС реального часу, яка складається з ядра, керування завданнями та часом, синхронізацією зв'язку (семафори, черга, тощо) і управління пам'яттю. Він створює умови для завдань працювати незалежно, не заважаючи одне одному, планування дозволяє завданням виконуватися вчасно і без помилок, тому полегшує проектування та додає можливість розширення додатків реального часу та процес проектування додатків значно скорочується у часі.

В управлінні пам’яттю, оскільки µClinux створено для процесорів без MMU, технологія управління віртуальною пам’яттю процесорів не використовується – застосовується лише реальну стратегію управління пам’яттю. Система використовує виділення пам'яті підкачки фізичної пам'яті під час запуску. Доступ до пам'яті – прямий та операційна система не захищає простір пам'яті.

Ядро Cortex-M4, що працює на частоті 168 МГц, знаходиться в STM32F4 серії, котра є провідною в STM32. Мікросхеми споживають енергію динамічно – до 230 мкА/МГц на максимально можливій частоті. 90-нм процес, прискорювач ART і динамічне масштабування потужності зменшують споживання струму. Саме це й поєднує безпрецедентну продуктивність, відмінну інтеграцію та привабливість ціноутворення [10].

Для порівняння STM32F4 серії мікроконтролерів, розлянемо рисунок 2.6

STM32F4 MCU Series
32-bit Arm® Cortex®-M4 – Up to 180 MHz

Product line	F_{CPU} (MHz)	Flash (KB)	RAM (KB)	Ethernet I/F IEEE 1588	2x CAN	Camera I/F	SDRAM I/F	Dual Quad-SPI	SAI	SPDIF RX	Chrom-ART Graphic Accelerator™	TFT LCD Controller	MIPI DSI
Advanced lines													
STM32F469[2]	180	512 K to 2056 K	384	•	•	•	•	•	•		•	•	•
STM32F429[2]	180	512 K to 2056 K	256	•	•	•	•		•		•	•	
STM32F427[2]	180	1024 K to 2056 K	256	•	•	•	•		•		•		
Foundation lines													
STM32F446	180	256 K to 512 K	128		•	•	•	•	•	•			
STM32F407[2]	168	512 K to 1024 K	192	•	•	•							
STM32F405[2]	168	512 K to 1024 K	192		•								

Рисунок 2.6 – Опис серії STM32F4 MCU

Для наших цілей достатньо використовувати Foundation лінійку. Питання постає між F407 та F446, проте зважаючи на пам’ять та необхідність додаткових технологій (SPI, I2C, I2S, USART, CAN, USB OTG, тощо) – вибір стає очевидним – STM32F407 Discovery підійде якнайкраще, деталі наведені на рисунку 2.7.

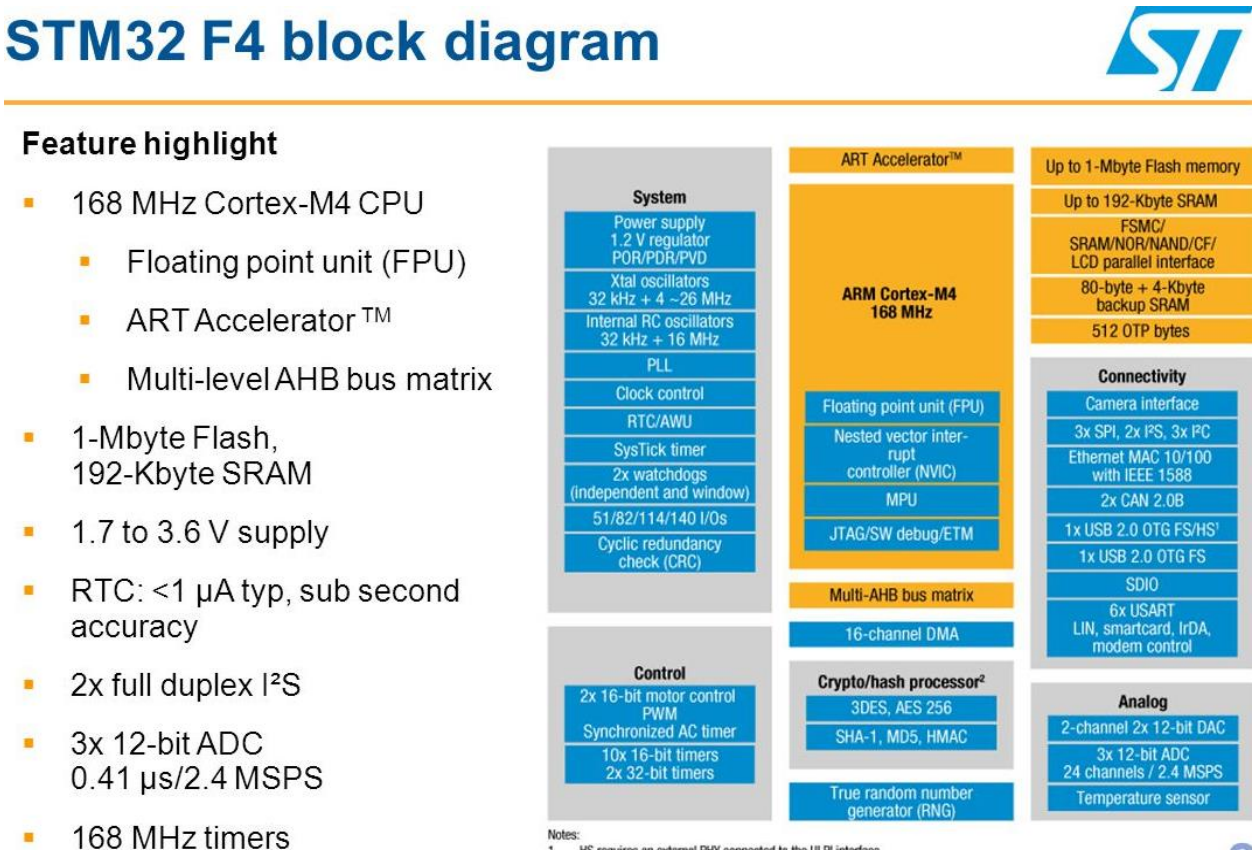


Рисунок 2.7 – Характеристика компонентів мікроконтролера STM32F4

Наявні 11 ліній, котрі сумісні із цифровим контролером сигналу (DSC) поєднують керування в режимі реального часу MCU з обробкою сигналів цифрового сигнального процесора (DSP). До того ж, включаючи багатий вибір передових периферійних пристроїв і достатньо великої пам'яті. Також, тут наявний модуль для роботи із числами з плаваючою комою – FPU, що значно розширює можливості обчислень як таких.

Широкі можливості підключення: STM32F407 мають Ethernet MAC10/100 – підтримка ормату IEEE 1588v2 і [8-14]-розрядний паралельний інтерфейс камери для підключення датчика камери CMOS [10].

- 2 x USB OTG (1 x USB HS)

- Аудіо PLL і 2 повнодуплексних I²S
- 15 комунікаційних інтерфейсів (6 x USART із швидкодією 11,25 Мбіт/с, 3 x SPI зі швидкістю до 45 Мбіт/с, 3 x I²C, 2 x CAN, SDIO)
- 2 x 12-розрядних ЦАП, 3 x 12-розрядних АЦП, які досягають 2,4 MSPS або 7,2 MSPS в режимі чергування
- 17 таймерів: 16- і 32-розрядні, що працюють на частоті до 168 МГц
- Аналоговий генератор випадкових чисел [11]

Також варто зазначити, що наявний розширюваний діапазон пам'яті із гнучким статичним контролером пам'яті, який підтримує Compact Flash, SRAM, PSRAM, NOR і NAND.

STM32F417 також інтегрує крипто/хеш-процесор, що забезпечує апаратне прискорення для AES 128, 192, 256, Triple DES і хеш (MD5, SHA-1).

Лінійка плат STM32F407 забезпечує від 512 Кбайт до 1 Мбайт Flash, 192 Кбайт SRAM і від 100 до 176 пінів, опис та використання котрих наведено на рисунку 2.8.

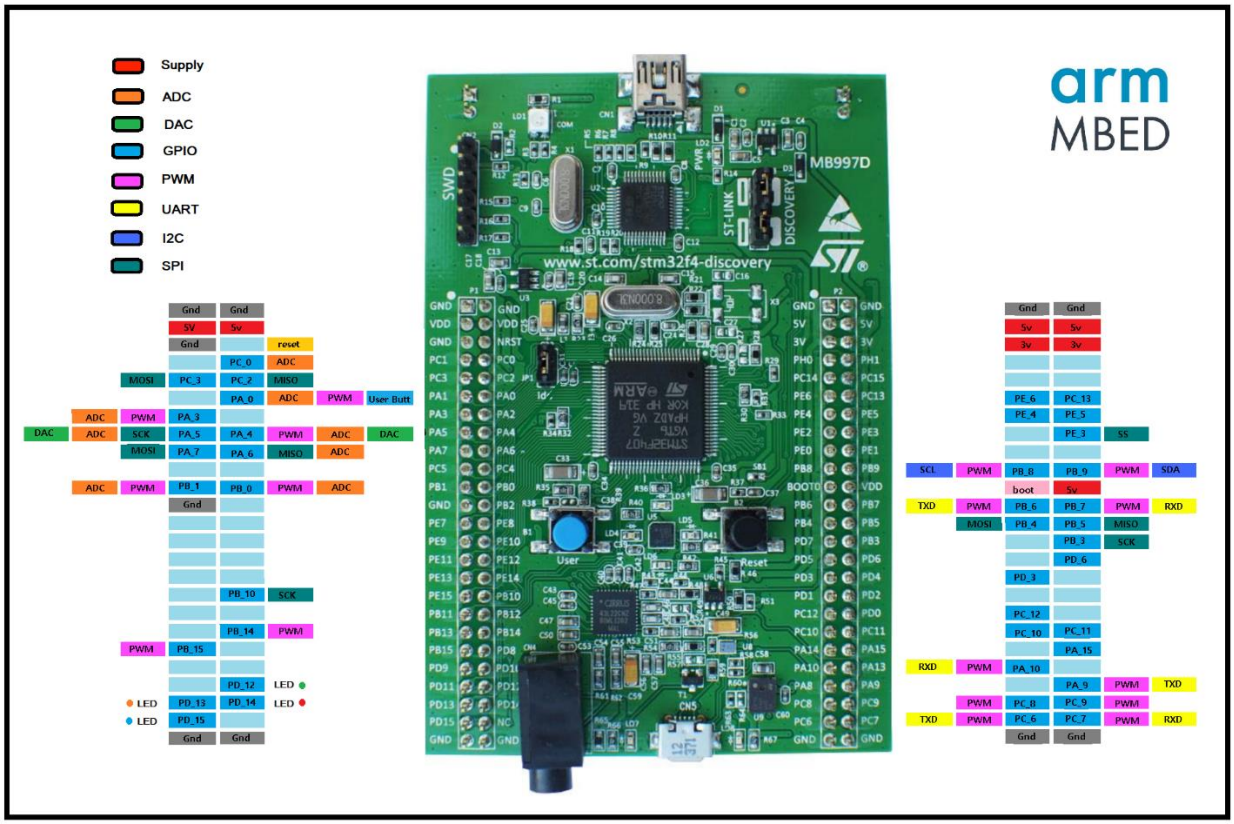


Рисунок 2.8 – Опис пінів GPIO на платах ESP32

Також, для відлагоджування програм, написаних для даної роботи знадобиться певний інтерфейс. Зручність полягає у тому, що у даному мікроконтролері наведено 2 інтерфейси відлагоджування на вибір розробника – JTAG та SWD. Відмінність полягає в тому, що SWD є 2-піновим, а JTAG 4-піновим. Програматор налагодження тут вказаний як ST-Link.

2.2 Вибір програмного забезпечення для проекту

Опісля того, як було обгрунтовано апаратні технології для даного проекту – варто розглянути й деякі аспекти в програмному забезпеченні, котрі можуть навіть відіграти ключову роль у подальшому дослідженні. Як мінімум, буде змога більш ефективно працювати із кодом, або використовувати ті чи інші переваги середовища для розробки та відлагодження програм.

2.2.1 Keil uVision

Спершу, необхідно розглянути таке середовище програмного забезпечення як Keil uVision. Його зовнішній вигляд можна оцінити, переглянувши зображення 2.9, де наведений графічний інтерфейс.

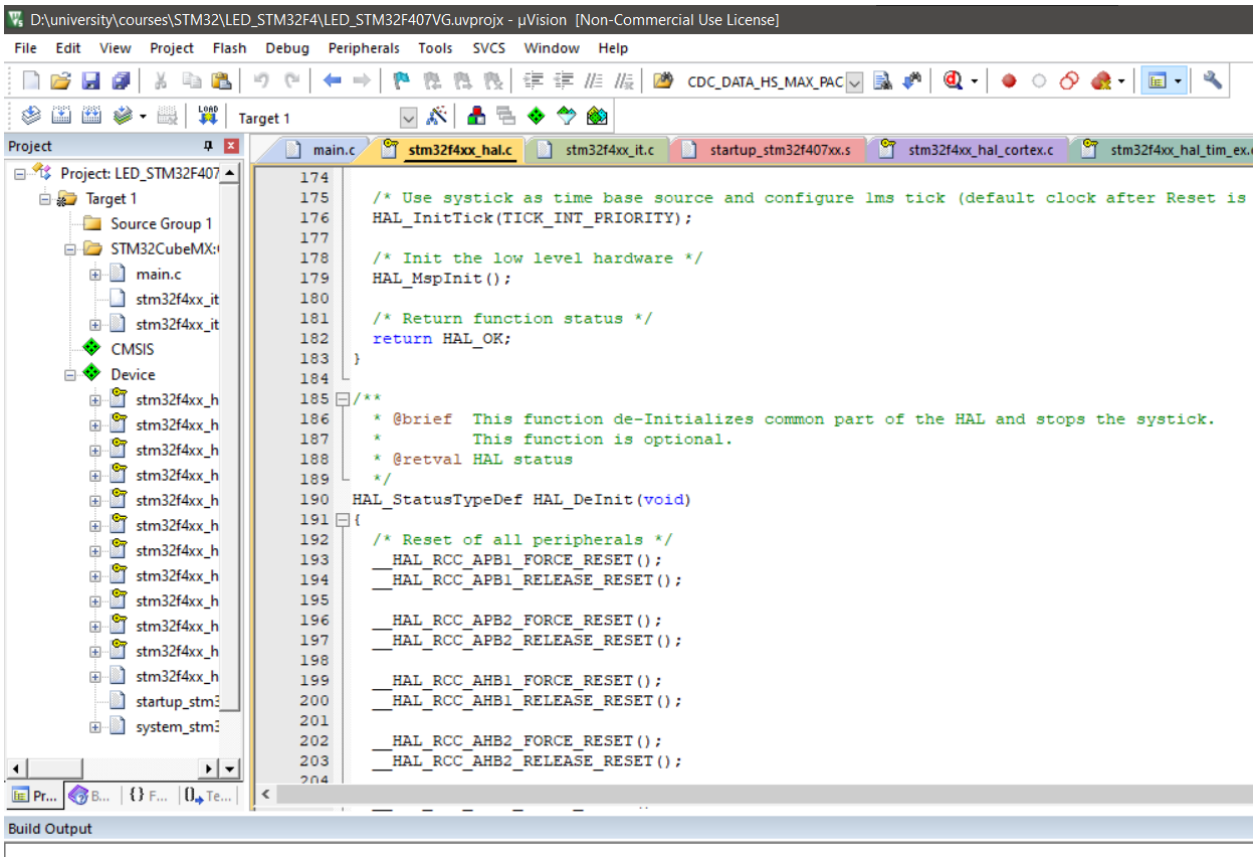


Рисунок 2.9 – Інтерфейс середовища Keil uVision v5.1

Як і в більшості графічних інтерфейсів середовищ розробки – Keil має інтуїтивне розміщення елементів вікон та найбільш часто використовувані функції – наведені у контекст меню.

Якщо брати до уваги засоби, котрі полегшують взаємодію із кодом, пропоную переглянути найбільш корисні із них.

- Embedded редактор для роботи із файлами безпосереднього програмування, як С наприклад. Відповідне забарвлення коду для візуалізації та автодоповнення за допомогою гарячих клавіш
- Менеджер проектів для ієрархії та з'єднання програмних подулів одного проекту
- База даних із версіями серій мікроконтролерів, у котрій наведена документація по кожному із них та наявні конфігураційні налаштування, за допомогою котрого відбувається автоматичне встановлення опцій
- Flash Tool – дозволяє запрограмувати Flash пам'ять мікроконтролера
- Симуляція відлагоджувача для мікропроцесора із віртуальною моделлю. Можливе відлагодження моделювання роботи ядра та різної периферії
- Аналізатор source коду
- Меню із наявними утилітами із зовнішнього простору

Із певних недоліків, є вартість ліцензії для некомерційного використання, котра є практично непідйомною в сучасних реаліях. Також, Keil uVersion не підтримується на Unix платформах – таких як Linux та MacOS.

2.2.2 STM32Cube IDE

Наступний кандидат відмінно складає конкуренцію попередньому продукту, оскільки окрім наведених вище корисних функцій – в ньому додані ще декілька. Вони користуються широким вжитком та суттєво спрощують розробку коду для мікроконтролерів STM32, як не дивно, бо ж сама компанія STM випустила даний продукт для embedded програмістів.

Спершу, розглянемо графічну частину – інтерфейс. Він, насправді, виглядає надто вже схоже до попереднього варіанту. І практично всі функції, описані вище – теж підтримує, зображено це на рисунку 2.10 нижче.

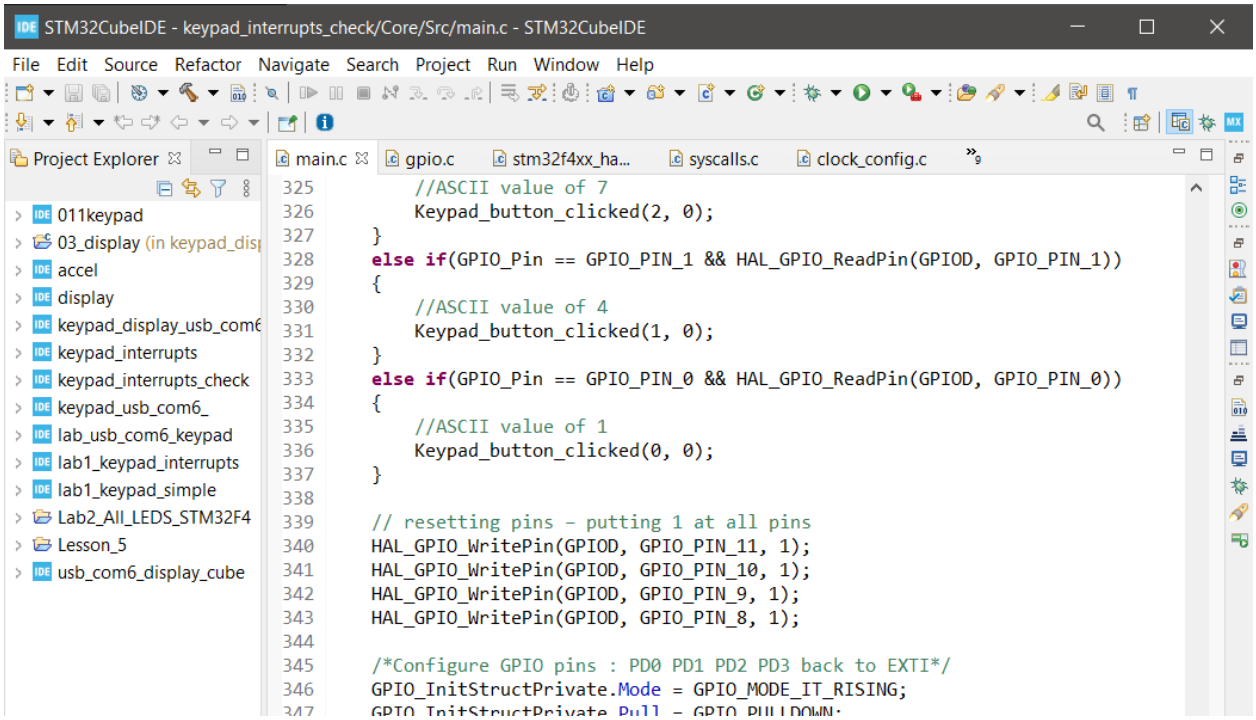


Рисунок 2.10 – Інтерфейс середовища STM32Cube IDE

Окрім вищесказаного, варто згадати зручності початкового конфігурування проекту на STM32Cube IDE. Як правило, почати варто із вибору необхідного мікроконтролера для проекту. Після вибору – є можливість налаштування схеми за допомогою графічного інтерфейсу, як зображено на рисунку 2.11. Тут немає необхідності писати початкову ініціалізацію пінів GPIO, встановлення інших параметрів периферії, тощо – задаючи значення певних бітів у відповідних регістрах вручну. Завдяки графічній надбудові від STM32CubeMX – є змога вибрати налаштування необхідного елемента та програма автоматично ініціалізує все за вас. Тобто, проаналізувавши вашу загальну структуру майбутнього проекту і оцінивши, налаштування котрих саме категорій елементів вам потрібне для роботи – ви налаштовуєте відповідні на Pinout View режими пінів та налаштовуєте їхні додаткові конфігурації зліва.

В додаток до вищесказаого, серед можливостей ще є можливості налаштування тактування системи, калькулятор витрат енергії для заданих параметрів, а також утиліта встановлення стеку Middleware, прикладом котрого може слугувати Ethernet, USB та інше.

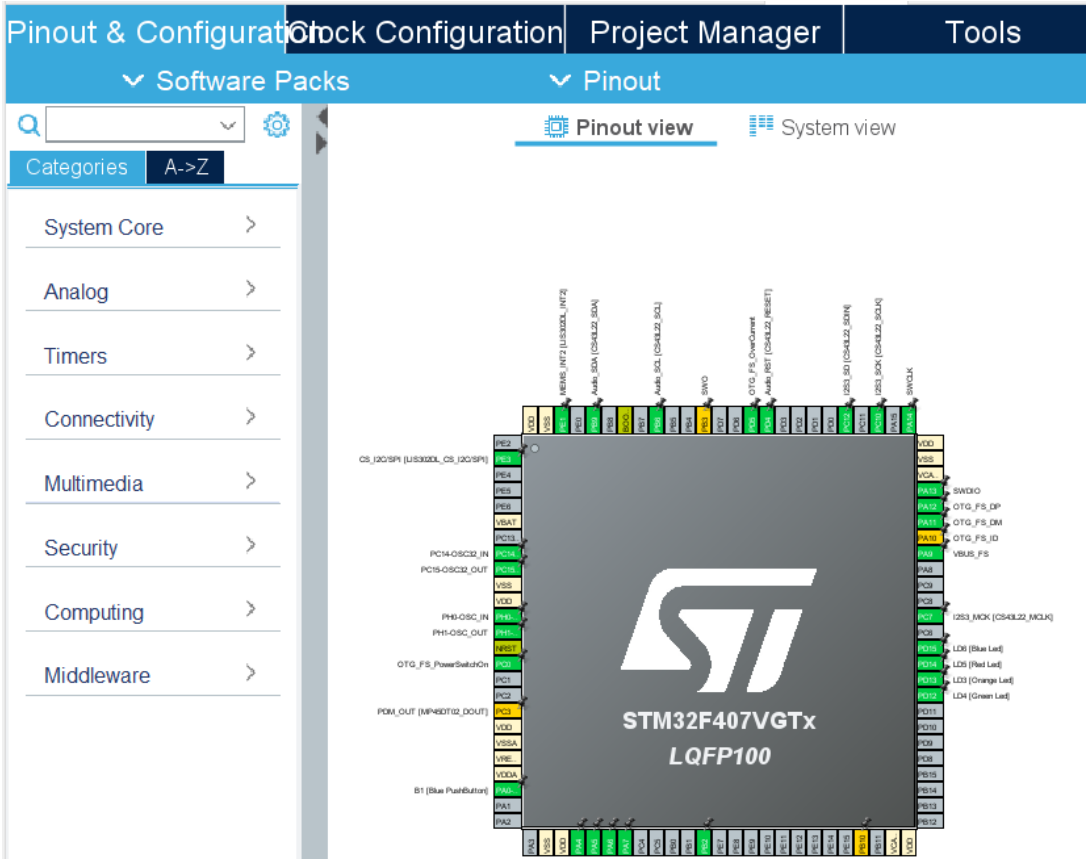


Рисунок 2.11 – Конфігурування всередині середовища STM32Cube IDE

Виконавши попередню підготовку, проініціалізвавши необхідну периферію та налаштувавши необхідне для шин тактування (дерево тактувань) – можна згенерувати код для проекту. Зручним є те, що опісля генерації є можливість зберегти файл конфігурування – в форматі .ioc. І якщо з'явиться потреба додати деяку периферію або змінити дерево конфігурацій, котре, до речі, є динамічно конфігураційним, то це буде надзвичайно легко. Оскільки саме із цього файлу STM32Cube згенерує актуальний код [11].

Також, говорячи про переваги, варто вказати ще й наявність можливості встановлення даного IDE не тільки для Windows ОС, але й для Linux та MacOS.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі була проведена проєктна розробка над оглядом технологій, котрі необхідні для створення проекту та загальну архітектурну структуру. Спершу, було проаналізовано різні варіанти плат.

Опісля детального розгляду переваг та недоліків кожного можливого варіанту, отриманих із документацій плат та порівняльних таблиць із характеристиками, як висновок, було обрано STM32F407DISCOVERY мікроконтролер.

Також, було надано обгрунтування вибору середовища програмування та засоби відлагодження для даного STM32 проекту. Серед головних претендентів фігурували Keil uVersion та STM32CubeIDE. Проте, зважаючи на порівняльні характеристики та зручність і швидкість написання програм у кожному із середовищ – значно вигравав другий - STM32CubeIDE. Саме він буде використаний для розробки драйвера в наступних розділах.

РОЗДІЛ 3
РОЗРОБКА ТА ОПИС АРХІТЕКТУРИ СИСТЕМИ

3.1 Застосовані технології, устаткування, розгортання програмного забезпечення

У відповідності із отриманими даними, отриманими в попередньому розділі, було вирішено обрати мікроконтролер STM32F407-DISCOVERY як «мозок» системи. Головна зручність, зважаючи на популярність даного продукту, є те, що коли з'явиться необхідність більш потужного МК – є багато варіацій вибору із STM32. Тобто, переконфігурація на інший пристрій даної компанії займе мінімальну кількість часу.

Також, для роботи із віртуальним COM портом буде потрібен драйвер компанії STM32. А для периферії буде задіяно клавіатуру 4x4 (як пристрій вводу). Окрім цього, решта обладнання, яке було використано в проєкті, зображено на рисунку 3.1. Детальна інформація опису того, як працюватиме периферія, буде наведена в наступних підрозділах.

Рисунок 3.1 – Мікроконтроллер та обладнання для системи

Для отримання даних із клавіатури, необхідно її під'єднати за допомогою дротів перемичок M2F до пінів GPIO мікроконтролера. Перевірка введених символів буде здійснена із допомогою консолі SWV. Проте, було б краще отримати введені дані безпосередньо до комп'ютера – тому, для цього використаймо Virtual COM Port. Він гарантуватиме передачу даних завдяки своєму протоколу, підключення пристрою із ПК буде через Micro-USB кабель. В свою чергу, mini-USB кабель буде використано для підключення плати до джерела живлення. Також, для перевірки підключення клавіатури та правильності налаштувань драйвера додамо дисплей WH1602B. Загальну структурну схему можна проаналізувати, проглянувши її у додатку 1.

3.2 Опис розробки

3.2.1 Взаємодія із клавіатурою на STM32 DISCOVERY

Спершу, потрібно зрозуміти принцип роботи клавіатури для STM32, потім можливість підключення її до плати та налаштувати для подальшої взаємодії клавіатури із дисплеєм.

Обладнання:

- Embedded Starter Kit
- Mini-USB кабель
- 1 клавіатура 4*4 (див. рис. 1)
- 8 дротів-перемичок M2F для зв'язку плати із клавіатурою , наведені на рисунку 3.2

Клавіатура 4x4 має 8 пінів для зчитування, за допомогою яких можна дізнатися, яка кнопка була натиснена. Перші чотири з них називаються row піни (рядкові). Друга четвірка - column піни (стовпцеві), котрі видно на рисунку 3.2

Побудувавши матрицю, можна зрозуміти, що кожен перетин рядка зі стовпцем утворюватиме кнопку клавіатури. Структурно, це ключ, який з'єднує рядок із стовпцем. Детально проглянути принципову схему можна у додатку 3.

Варто зазначити, що стовпцеві піни – C1, C2, C3, C4 – піни входу для мікроконтролера (input), а рядки – R1, R2, R3, R4 піни виходу (output).

Рисунок 3.2 - Клавіатура 4*4 та дроти-перемички M2F

Коли жодна кнопка не натиснута – рядки та стовпці ізольовані і немає жодного зв’язку між ними (значення 1 у регістрі). Наприклад, розглянемо натиснення кнопки ‘1’. При натиску на неї, R1 та C1 з’єднується ключем (значення 0) і це потрібно програмно виявити. Деталі характеристик наведені на рисунках 3.3 та 3.4.

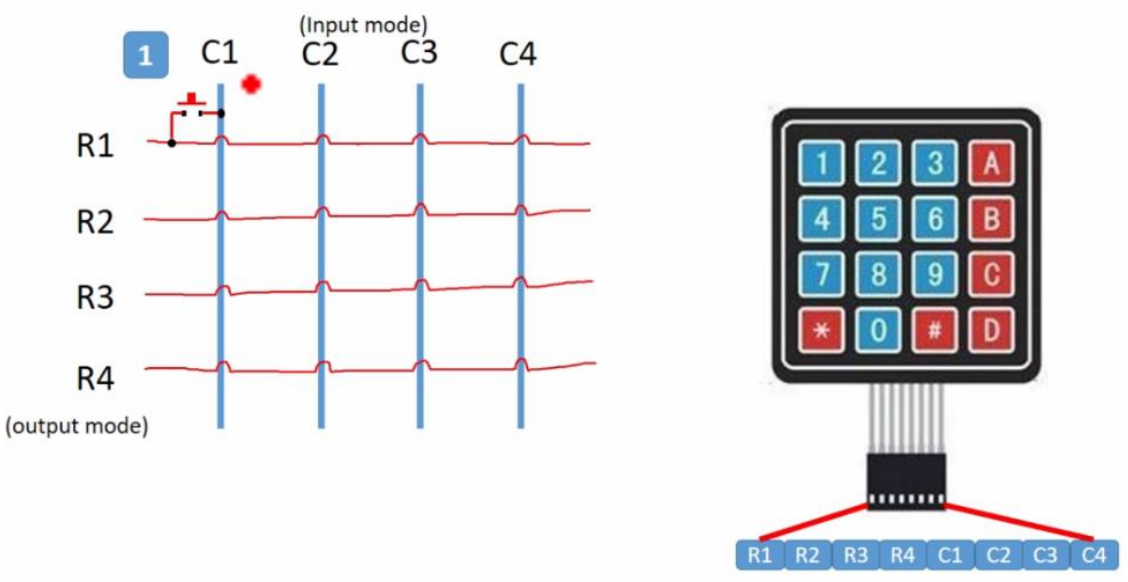


Рисунок 3.3 - Клавіатура та матриця пінів

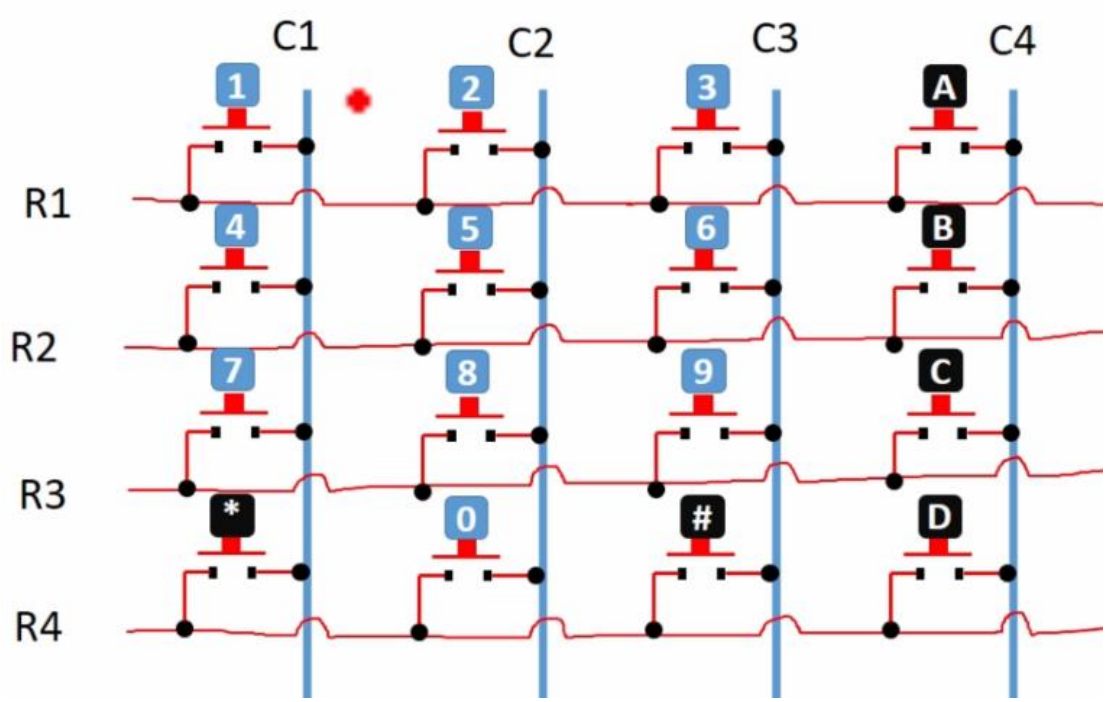


Рисунок 3.4 - Клавіатура та матриця пінів

Тепер, необхідно розглянути рисунок 3.5, де схематично зображене підключення клавіатури до STM32F4. Для реалізації стовпцевих пінів використаємо **PD8 – PD11** піни в режимі входу, а для рядкових – **PD0-PD3** піни в режимі виходу.

Також, для стовпцевих пінів нам знадобляться pull-up резистори. Вони забезпечать чітко визначену напругу (тобто VCC або логічний максимум) в колі, коли ключ відкритий. Інакше – пін входу може сприймати випадковий сигнал кола і може коливатися між 1 і 0. Це зветься плаваючим піном. Ці резистори – внутрішні – тобто, наявні всередині мікроконтроллера і не потрібно буде підключати їх ззовні. Знаходяться вони посередині між піном та VCC(+). Кнопка не натиснута – High (логічна 1, Vcc), а якщо натиснута – Low (логічний 0, GND).

Отже, коли потрібно програмно визначити, яка кнопка зараз натиснута – починається перевірка кожного ряду.

На першій ітерації циклу R1 встановлюється в 0 (Low), а інші рядкові піни мають значення 1. Перевіряється C_i (i=1..4). Якщо наявне значення C_i = 0, отже кнопка R1 рядка та C_i стовпця натиснута. Якщо не знайдена, йдемо на наступну ітерацію.

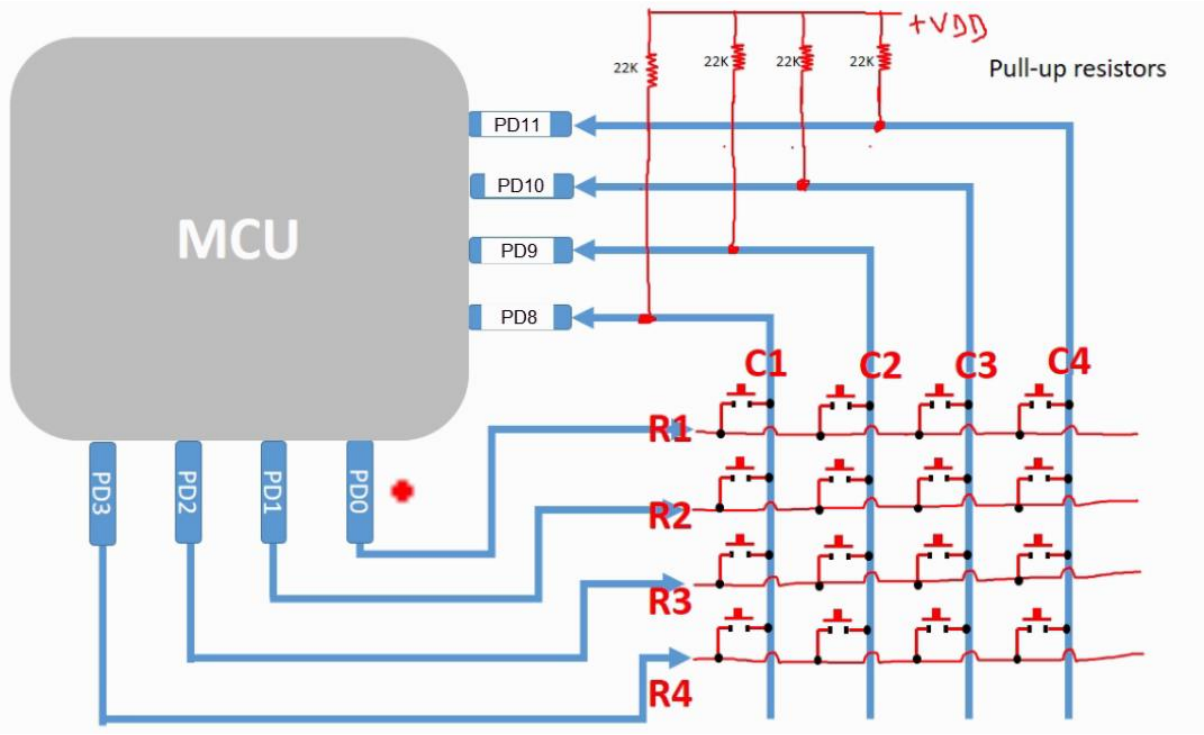


Рисунок 3.5 – Підключення клавіатури до STM32F4

3.2.2 Алгоритм імплементації

Спершу, потрібно зрозуміти, як програмно створити алгоритм дій, котрий гарантовано видасть необхідний нам результат:

1. Визначити які IO піни використовуватимуться для реалізації R_i, C_i, де i=1..4 (**PD8 – PD11** піни в режимі входу (стовпці), **PD0-PD3** піни в режимі виходу (рядки)
2. Створити змінні-вказівники для обробки регістрів, відображених в пам’яті
3. Ініціалізувати змінні-вказівники з адресами регістрів
4. Ініціалізація:
    - Встановити всі R_i – в output режим
    - Встановити всі C_i – в input режим
    - Активувати внутрішні pull-up резистори для C_i
5. Імплементація визначення кнопки

Детальний алгоритм імплементації наведено у додатку 4.

3.2.3 Адреси необхідних регістрів STM32F4

Діапазон адрес для STM32F4xx GPIOD регістра: 0x4002 0C00 – 0x4002 0FFF [12]. Налаштування режиму роботи пінів порту GPIOD:

PD0 – PD3 – у режимі виходу (output), отже значення бітів 01, дивитися рисунок 3.7. PD8 – PD11 – у режимі входу (input), значення бітів 00 (по замовчуванню).

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

Рисунок 3.6 – Конфігураційні біти для регістру GPIO порту

Налаштування для стовпцевих пінів PD8 – PD11 pull-up резисторів. Встановлення відповідних бітів до регістру – 01, дивитися на рисунок 3.7

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

Рисунок 3.7 – Конфігураційні біти для pull-up регістру GPIO

Також, налаштування для стовпців PD8 – PD11- регістру вхідних даних, а для PD0 – PD3 - регістру вихідних даних. Потрібні для визначення натиснутої клавіші.

Режим входу надає доступ до регістру вводу даних порту – GPIOx_IDR, а режим виводу – до регістру виводу даних порту, відповідно GPIOx_ODR. Перший регістр надає доступ тільки для зчитування бітів стану IDR0 – IDR15, а другий – ще й запис до бітів стану ODR0 – ODR15. Деталі налаштування дивитися на рисунку 3.8 [12].

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Рисунок 3.8 – Конфігураційні біти вводу й виводу регістру GPIO

Також, не слід забувати про підключення годинника для периферії на STM32F4 – RCC AHB1. Оскільки, використовуємо GPIOD порт, то переглянувши документацію (дивитися рисунок 3.9) – для цього потрібно виставити 3-ій біт [12]. Всі деталі стосовно функціональної схеми можна переглянути у додатку 2.

7.3.10 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw			rw	rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 3.9 – Підключення периферійного годинника для GPIOD

3.2.4 Зовнішні переривання

Також, для програмування відслідковування натиснутих клавіш клавіатури Зовнішні переривання – переривання, які відбуваються тоді, коли надходить сигнал від зовнішніх пристроїв. Саме зовнішні переривання дозволяють оперативно обробляти сигнали зовнішніх пристроїв, наприклад реагувати на натискання кнопки, сигнали датчиків, і так далі.

Контролери STM32 мають у своєму складі контролер зовнішніх переривань EXTI. Загалом у мікроконтролерів серії STM32F4 зовнішні переривання EXTI0-EXTI15 налаштовують для роботи від зміни рівня на відповідній ніжці контролера. У контролерах STM32 ми самі можемо вибрати ніжки, на які будуть налаштовані переривання EXTI, розглянуто на рисунку 3.10 нижче.

Тут видно 16 ліній, підключених через мультиплексори до однойменних пінів всіх портів. Тобто одночасно ми можемо обробити 16 ніжок контролера, але, як видно з мультиплексної організації, всі вони мають бути з різними

номерами портів. Тобто ми можемо обробити одночасно ніжки PA1 та PC2, але не можемо обробити PA1 та PC1.

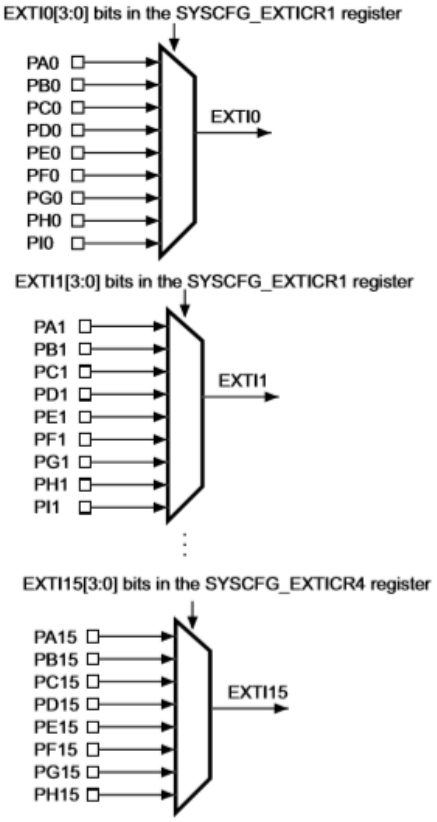


Рисунок 3.10 - Зовнішня мапа переривань для STM32F407

Для переривання EXTI0 ми можемо вибрати нульовий порт біт (PA0, PB0, PC0 і так далі), для EXTI1 перший біт, і так далі. Сім інших переривань EXTI16-EXTI22, підключені в наступному порядку:

- EXTI Лінія16 підключена до PVD output
- EXTI Лінія16 підключена до RTC Alarm події
- EXTI Лінія16 підключена до USB OTG FS Wakeup події
- EXTI Лінія16 підключена до Ethernet Wakeup події
- EXTI Лінія16 підключена до USB OTG HS Wakeup події
- EXTI Лінія16 підключена до RTC Tamper / Timestamp подій
- EXTI Лінія16 підключена до RTC Wakeup події [13]

Тобто ми можемо ще обробити зовнішні події від програмованого детектора напруги, від RTC, від "пробуджень" USB і Ethernet і т.д.

Також, STM32F4 має 7 обробників переривань для виводів GPIO. Їхні характеристики наведені в таблиці 3.1.

Таблиця 3.1 – Обробники переривань в STM32F407-DISCOVERY

Irq	Handler	Description
EXTI0_IRQn	EXTI0_IRQHandler	Handler for pins connected to line 0
EXTI1_IRQn	EXTI1_IRQHandler	Handler for pins connected to line 1
EXTI2_IRQn	EXTI2_IRQHandler	Handler for pins connected to line 2
EXTI3_IRQn	EXTI3_IRQHandler	Handler for pins connected to line 3
EXTI4_IRQn	EXTI4_IRQHandler	Handler for pins connected to line 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	Handler for pins connected to line 5 to 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	Handler for pins connected to line 10 to 15

У цій таблиці показано, який IRQ (Interrupt ReQuest) необхідно встановити для NVIC (Nested Vector Interrupt Controller) (перший стовпець) та імена функцій для обробки переривань (другий стовпець). Також можна помітити, що лише рядки від 0 до 4 мають власний обробник IRQ. Рядки 5-9 мають однаковий обробник переривань, це справедливо й для 10-15.

3.2.5 Взаємодія з віртуальним COM портом на STM32 DISCOVERY

Спершу, необхідно зрозуміти принцип роботи COM порта для STM32,підключення плати до ПК відбувається за допомогою micro-USB кабелю

COM-порт - це інтерфейс вводу-виводу, який дозволяє підключити послідовний пристрій до комп'ютера. Також, COM-порти називаються послідовними портами. COM порт використовує асинхронний інтерфейс, який може передавати/отримувати один біт даних за раз при підключенні до послідовного пристрою. Це забезпечує простий і надійний спосіб зв’язку.

Більшість сучасних комп’ютерів не обладнані COM-портами, але є багато пристроїв із послідовним портом, які все ще використовуються. Це лабораторні інструменти, медичне обладнання та системи торгових точок, які досі часто використовують послідовне підключення, дивитися рисунок 3.11.

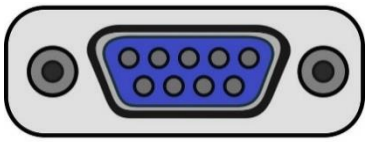


Рисунок 3.11 - COM порт

Отже, послідовні порти не є стандартним обладнанням більшості сучасних ПК. Зараз вони, як правило, замінюються одним або кількома інтерфейсами USB (*Universal Serial Bus*) - універсальна послідовна шина, стандарт роз'ємів і кабелів для передачі даних (до 40 Гбіт/с) та живлення невеликих пристроїв. Тобто, ви можете використати адаптер USB-SERIAL, який може забезпечити один або кілька COM-портів для машин, які не мають встановлених послідовних інтерфейсів. Існує багато доступних рішень, і більшість з них недорогі та прості у реалізації.

Ми ж використаємо віртуальний COM-порт, який наданий для нашої плати компанією STM. Віртуальний послідовний порт — це програмне уявлення послідовного порту, який або не підключається до реального послідовного порту, або додає функціональність справжньому послідовному порту за допомогою програмного розширення. Тобто Virtual serial port на основі програмного забезпечення представляє один або кілька ідентифікаторів віртуального послідовного порту на ПК, які інші програми можуть бачити та взаємодіяти з ними, як ніби вони були реальними апаратними портами. Але дані, надіслані та отримані на ці віртуальні пристрої, обробляються програмним забезпеченням, яке маніпулює переданими та отриманими даними для надання більшої функціональності.

Операційні системи зазвичай не забезпечують можливості віртуального послідовного порту. Цю можливість можуть додати сторонні додатки, як у нашому випадку компанія STM.

Послідовний порт, як правило, може відстежуватися або передаватися лише одним пристроєм за один раз відповідно до обмежень більшості OS, але програма віртуального послідовного порту може створити два віртуальні порти, що дозволяє двом окремим програмам відстежувати одні й ті ж дані.

Наприклад, пристрій GPS, який виводить дані про місцезнаходження на послідовний порт ПК, може бути цікавим для кількох програм одночасно.

Інший варіант — зв’язуватися з іншим serial пристроєм через Інтернет або локальну мережу, як ніби вони були локально підключені, використовуючи serial порт через локальну мережу. Це дозволяє програмному забезпеченню, призначеному для взаємодії з пристроєм через локальний фізичний послідовний порт, замість цього спілкуватися на великій відстані.

Також, поширення використання для Bluetooth. Він реалізує віртуальні послідовні порти через профіль serial порту. Наприклад, це стандартний спосіб отримання даних від GPS-модулів, обладнаних Bluetooth.

3.2.6 USB конфігурації

USB On-The-Go, USB OTG або OTG – це специфікація інтерфейсу USB, яка дозволяє USB-пристрою, наприклад мобільному телефону, бути хостом USB, дозволяючи підключення та використання заразом інших USB - пристроїв, таких як USB-флеш-накопичувач, мишка чи клавіатура (декілька пристроїв можна підключити через USB-хаб). Використання USB OTG дозволяє пристроям перемикатися між ролями хоста і кінцевого (периферійного) пристрою. Наприклад, смартфон може зчитувати дані зі знімного носія як хост, але при підключенні до хост-комп'ютера брати на себе роль пристрою (USB-пристрій) [14].

Можливі 2 конфігураційні режими USB – Пристрій (Device) або Хост (Host). Хост ініціює весь прийом-передачу даних по шині, а також надає живлення до підключеного девайсу - пристрою. А функція пристрою відповідати на запити хоста. В даному випадку, хостом виступатиме ПК, а пристроєм буде плата STM32 [15].

Для підключення плати до ПК через USB, потрібно створити віртуальний COM порт. Для передачі даних використаємо CDC (Communication Device Class) – клас пристроїв middleware, який дасть змогу передавати чи приймати дані між пристроєм та хостом.

Також, для обміну даними необхідно приєднати плату STM32 до ПК за допомогою micro-USB кабелю (див. рис. 2). Підключення відбудеться до шини USB OTG FS плати STM32F407 Discovery.

Є 2 способи використання USB OTG – з FS(full-speed) або HS(high-speed) контролерами. У проєкті використаємо USB OTG FS, оскільки швидкості передачі буде більш, ніж достатньо і не потрібне додаткове налаштування зовнішнього PHY (physical layer) – фізичного шару для використання ULPI трансивера. У FS контролера є вбудований on-chip PHY. Як правило, сенс підключати USB OTG HS контролер є, коли необхідне використання DMA (direct memory access). Детальніше, пропоную поглянути на таблицю 3.2 [16, 17].

Таблиця 3.2 - Порівняльна характеристика OTG : FS та HS

Функція	**OTG_FS**	**OTG_HS**
Підтримувані стандартні швидкості пристрою USB	FS (full-speed)	FS (full-speed), HS high-speed)
Підтримувані стандартні швидкості хосту USB	LS (low-speed), FS (full-speed)	LS (low-speed), FS (full-speed), HS (high-speed)
Кінцеві точки пристрою USB	0 .. 3	0 .. 5
Канали хосту USB	0 .. 7	0 .. 11
Підтримка DMA	-	Наявна
Додатковий зовнішній інтерфейс ULPI PHY	-	Наявний

Щоб підключитися через COM port до комп’ютера – мікроконтролери компанії STM32 використовують micro-USB кабель, так, як наведено на рисунку 3.12 нижче.

Рисунок 3.12 - Схема підключення mini- та micro-USB

3.3 Створення проекту в STM32 CubeIDE

Спершу необхідно відкрити STM32 CubeIDE. Для створення нового проекту – виберемо File -> New -> STM32 Project та введемо дану модель плати, як наведено на рисунку 3.13.

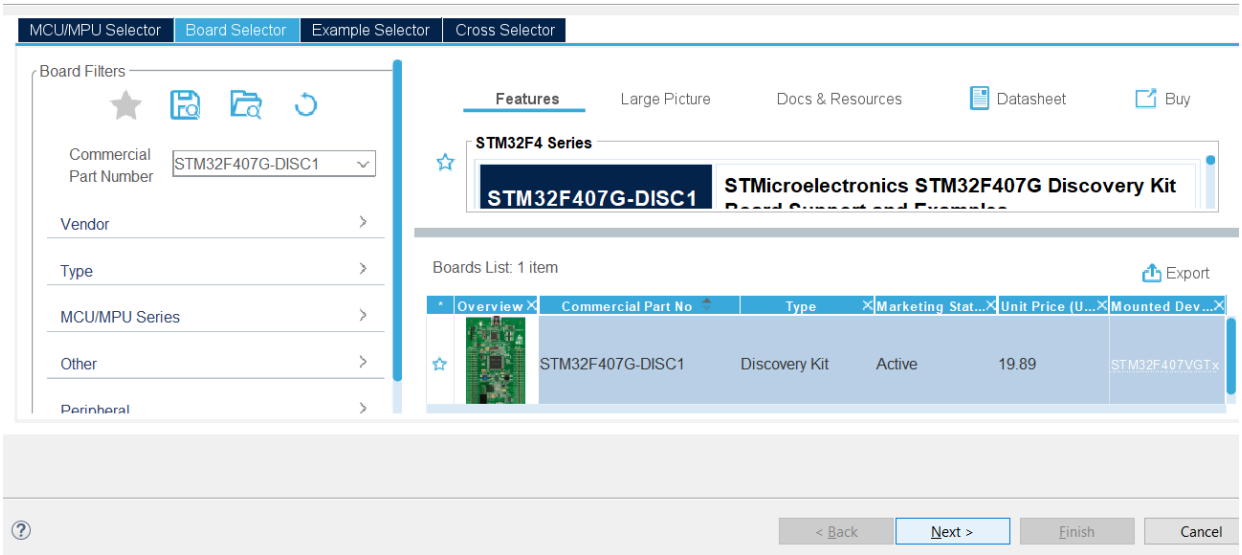


Рисунок. 3.13 - Вибір плати для проекту

Наступним кроком, є створення проекту STM32, мовою програмування виберемо C. Бінарний тип файлу вказуємо – виконуван ий (executable) файл. Оскільки спершу варто написати простий проект-драйвер для клавіатури без використання переривань – створимо порожній проект без запропонованих програмою значень всіх пінів по замовчуванню, пізніше самі встановимо необхідні біти на регістрах. Приклад створення наведено на рисунку 3.14.

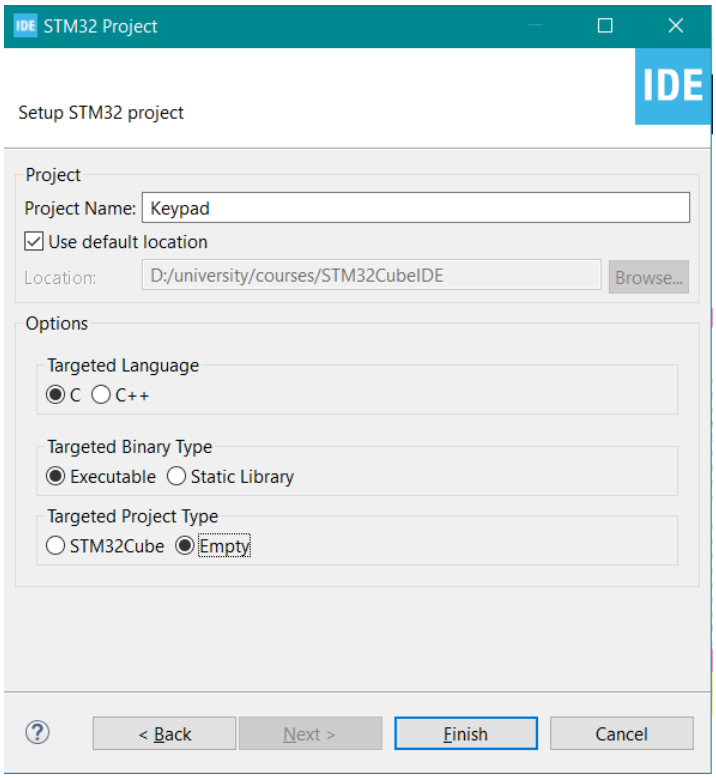


Рисунок. 3.14 - Вибір конфігурації для проекту

Створюємо файли для роботи клавіатури - keypad.h та keypad.c. Для цього спершу необхідно визначити вказівники на потрібні регістри для роботи - про які йшла мова у пункті 3.2.3 вище. Їхні адреси вказані на рисунку 3.15.

```
//peripheral register addresses
uint32_t volatile *const pGPIODModeReg  =  (uint32_t*)(0x40020C00);
uint32_t volatile *const pInPutDataReg  =  (uint32_t*)(0x40020C00+0x10);
uint32_t volatile *const pOutPutDataReg =  (uint32_t*)(0x40020C00+0x14);
uint32_t volatile *const pClockCtrlReg  =  (uint32_t*)(0x40023800+0x30);
uint32_t volatile *const pPullupDownReg =  (uint32_t*)(0x40020C00 + 0x0C);
```

Рисунок. 3.15 - Вибір конфігурації для проекту

Спочатку, перед тим, як використовувати клавіатуру – потрібно задати її початковий стан – ініціалізувати для відповідних регістрів стани необхідних пінів, котрі використовуватимуться для рядкових та стовпцевих пінів у GPIOD. Детальний опис зміни бітів у наведених вище регістрів порядково – наведено на. Число 0x55 – представлення послідовності 0101 0101 (бінарний) у шістнадцятковій системі числення. За аналогією вище, шістнадцятковий код 0xFF – 1111 1111 у бінарному вигляді.

```
void Setup_GPIO_for_keypad() {
   //1.Enable the peripheral clock of GPIOD peripheral
      *pClockCtrlReg |= ( 1 << 3);
  // 2.configure PD0,PD1,PD2,PD3 as output (rows)
      *pGPIODModeReg &= ~(0xFF); //clear
      *pGPIODModeReg |= 0x55;   //set
      // 3. configure PD8 , PD9, PD10, PD11 as input(columns)
  *pGPIODModeReg &= ~(0xFF << 16);
      // 4.Enable internal pull-up resistors for PD8 PD9
      // PD10 PD11
  *pPullupDownReg &= ~(0xFF << 16);
  *pPullupDownReg |=  (0x55 << 16);
}
```

Коли успішно відбулася ініціалізація GPIO D конфігурації, де нас цікавили PD8, PD9, PD10, PD11 піни (колонки) та PD0, PD1, PD2, PD3 піни (рядки) – тепер, варто зупинитися на детальному описі алгоритму відловлення натиснутої кнопки.

У циклі for для кожного i-того рядка Ri – проскануємо всі чотири колонки. Спершу, виставимо в pOutputDataReg – 1111, що означатиме - всі рядки неактивні (нагадаю, що активний ключ в даному випадку становить 0), для того, щоб обнулити попередній стан. Потім, виставляємо i-тий рядок (саме той, що нам потрібен) в 0 за допомогою інверсивної техніки pOutputDataReg &= ~(1 << i). Далі, здійснюємо перевірку на pInputDataReg GPIOD. Оскільки колонкові піни займають позиції від 8 до 11 – з даного вказівника нас цікавлять розіменовані біти відповідного порядку, котрі можна визначити за допомогою операцій pOutputDataReg & (1 << j), де j = [8, 11]. Якщо котрийсь із колонкових пінів задовільняє умові – його значення рівне 0 – викликаємо функцію printf, котрій вкажемо назву клавіші. Детальніше – код можна переглянути нижче, де наведені повні функціональні рядки.

Також, необхідно врахувати той факт, що без переривань потрібна затримка delay у перевірках, оскільки тоді одна й та ж умова спрацьовуватиме декілька разів, бо операції виконуються із великою швидкістю. Тому, варто їх

додати у перевірку кожної колонки (тоді, достатньо буде невеликої – мілісекунди) або тільки одну – на кожну ітерацію рядка.

```
void Keypad_loop() {
      for (uint8_t i=0; i<4; ++i) {
            //make all rows HIGH
            *pOutPutDataReg |= 0x0f;

            //make Ri LOW(PD0)
            *pOutPutDataReg &= ~( 1 << i);

            //scan the columns
            //check C1 of Ri is low or high
            if(!(*pInPutDataReg & ( 1 << 8))) {
                  //key is pressed

                  //delay for only 1 char printed
                  Delay();
                  printf("%s", keypad_symbols[i*4+0]);
            }
            //check C2 of Ri is low or high
            if(!(*pInPutDataReg & ( 1 << 9))) {
                  Delay();
                  printf("%s", keypad_symbols[i*4+1]);

            }
            //check C3 of Ri is low or high
            if(!(*pInPutDataReg & ( 1 << 10))) {
                  Delay();
                  printf("%s", keypad_symbols[i*4+2]);
            }
            //check C4 of Ri is low or high
            if(!(*pInPutDataReg & ( 1 << 11))) {
                  Delay();
                  printf("%s", keypad_symbols[i*4+3]);
            }
            Delay();
      }
}
```

Тепер, коли код для драйвера клавіатури написаний, змінюємо вміст файлів syscalls.c (для printf в Serial Wire View - SWV). Вивід printf() викликає функцію _write (файл ‘‘syscalls.c’’), яка виводить символи в консоль [18].

Створимо функцію ITM_SendChar() для передачі символа пакетом по шині (trace bus). Додамо її на початок файлу syscalls.c , як наведено в коді нижче.

```
//Debug Exception and Monitor Control Register base address
#define DEMCR                 *((volatile uint32_t*) 0xE000EDFCU )
/* ITM register addresses */
#define ITM_STIMULUS_PORT0    *((volatile uint32_t*) 0xE0000000 )
```

```
#define ITM_TRACE_EN          *((volatile uint32_t*) 0xE0000E00 )
void ITM_SendChar(uint8_t ch)
{
      //Enable TRCENA
      DEMCR |= ( 1 << 24);
      //enable stimulus port 0
      ITM_TRACE_EN |= ( 1 << 0);
      // read FIFO status in bit [0]:
      while(!(ITM_STIMULUS_PORT0 & 1));
      //Write to ITM stimulus port0
      ITM_STIMULUS_PORT0 = ch;
}
```

Опісля, відредагуємо функцію _write, замість дефолтної __io_putchar() ми будемо викликати ITM_SendChar(), як наведенона рисунку 3.16.

```
136  __attribute__((weak)) int _write(int file, char *ptr, int len)
137  {
138      int DataIdx;
139
140      for (DataIdx = 0; DataIdx < len; DataIdx++)
141      {
142          //__io_putchar(*ptr++);
143          ITM_SendChar(*ptr++);
144      }
145      return len;
146  }
```

Рисунок. 3.16 – Редагування функції для запису символів

3.4 Використання переривань в проекті в STM32CubeIDE

За аналогією пункту 3.3. виберемо налаштування проекту:

- File -> New -> STM32 Project.
- Board Selector -> STM32F407G-DISC1 -> Next
- Targeted Project Type -> STM32Cube - > Finish

Спершу, налаштуємо SYS Mode – так, як показано на рисунку 3.17.

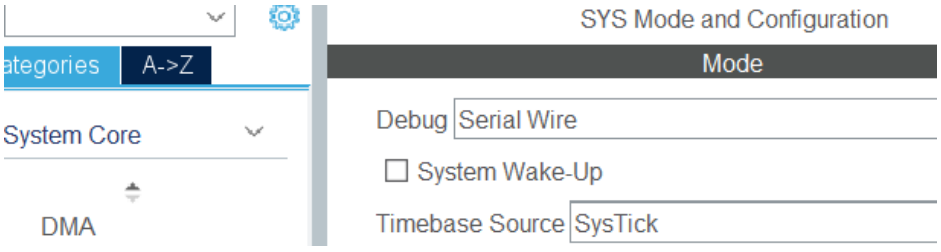


Рисунок 3.17 – Налаштування SYS Debug в STM32 проекті

Також, необхідно визначити RCC – котрий відповідає за контроль внутрішньої периферії, в тому числі reset сигнали та розподіл годинника. Встановимо HSE – високошвидкісний годинник – кристальний резонатор, як зображено на рисунку 3.18.

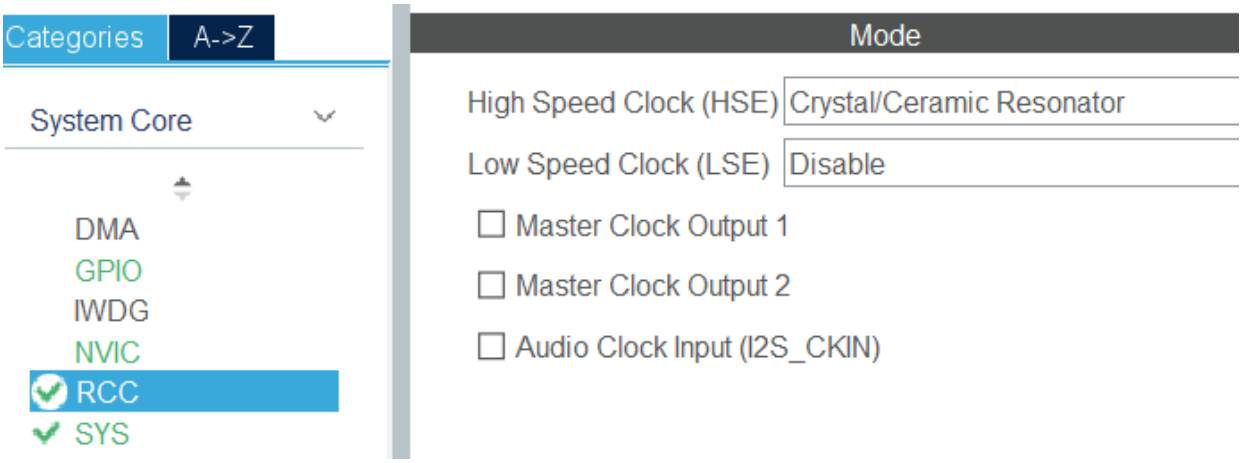


Рис.3.18 – Налаштування RCC HSE в STM32 проекті

Тепер, налаштуємо у відповідні режими піни для роботи із клавіатурою. Рядкові піни (PD0 - PD3) встановити у режим GPIO_EXTI. Стовпцеві піни (PD8 – PD11) встановити у режим GPIO_Output (див. рис. нижче).

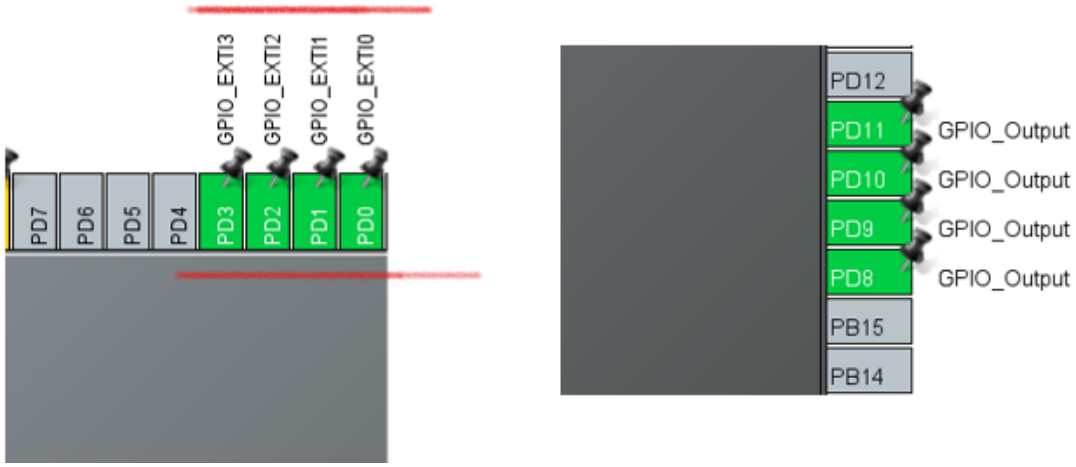


Рис.3.19 – Налаштування GPIO пінів в STM32 проекті

Далі, потрібно ввімкнути лінії переривань для рядкових пінів у NVIC полі налаштувань (Nested Vectored Interrupt Controller), тобто EXTI line 0-3 interrupts, як показано на рисунку 3.20.

Оскільки, тепер налаштування проекту завершено – можна згенерувати код проекту у STM32 CubeIDE.

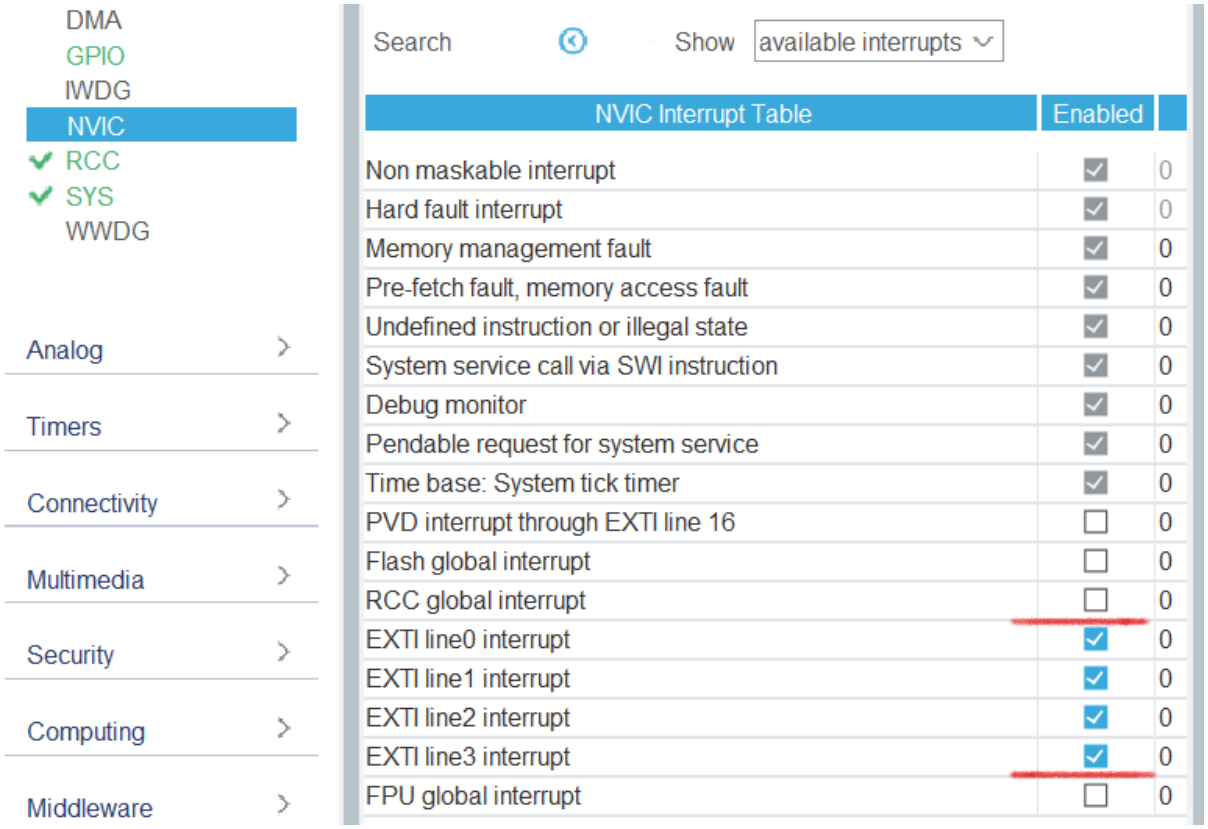


Рисунок 3.20 – Налаштування NVIC таблиці переривань в проекті

Для початку, змінимо код у файлі main.c для роботи із клавіатурою, адаптувавши її для використання переривань, замість простого режиму поллінгу.

Спершу, варто винести код виводу на дисплей в окрему спеціалізовану для цього функцію Keypad_button_clicked, як наведено на рисунку 3.21.

```
void Keypad_button_clicked( int row, int column) {
    Print_display(keypad_symbols[row*4+column]);
    data[n_chars] = keypad_symbols[row*4+column];
    ++n_chars;
}
```

Рисунок 3.21 – Налаштування NVIC таблиці переривань в проекті

Саму обробку переривань помістимо в функцію HAL_GPIO_EXTI_Callback(uin16_t GPIO_Pin), котра розрахована на переривання певного піна GPIO – в нашому випадку це GPIOD.

Починається обробка колбеку тим, що відбувається встановлення рядкових пінів PD0 – PD3 в режим вводу без pull-up регістру та ініціалізації GPIOD в цілому. Виконавши це, встановлюємо для PD11 – значення 1, а всі

решта стовпцевих пінів в 0. Далі, за допомогою перебору умов кожного рядкового піна даного стовпця (четвертого, PD11, де розміщені клавіші із символами A, B, C, D) – визначаємо котра із клавіш викликала обробник переривань. Коли умова ствердилася – викликаємо відповідний для кнопки функціонал – в моєму випадку, це виведення символу на дисплей або кнопка конвертації введеної послідовності символів із шістнадцяткової системи числення до бінарного виду (двійкова система числення). Шматок функції – обробника можна подивитися нижче, де наведена перевірка лише для одного стовпця, для якого вище було описано.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  currentMillis = HAL_GetTick();
  if (currentMillis - previousMillis > 500) {
       /*Configure GPIO pins : PD0 PD1 PD2 PD3 to GPIO_INPUT*/
    GPIO_InitStructPrivate.Pin =
GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0;
    GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
    GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
    GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStructPrivate);
    // checking for last 4th column - PD11=1, others - 0
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 1);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);

    if(GPIO_Pin == GPIO_PIN_3 && HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_3))
    {
     //ASCII value of D
      Display_Write_Ins(GO_TO_START_SECOND_LINE);
       Convert_hex_to_bin();
    }
    else if(GPIO_Pin == GPIO_PIN_2 && HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_2))
    {
     Keypad_button_clicked(2, 3); //ASCII value of C
    }
    else if(GPIO_Pin == GPIO_PIN_1 && HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_1))
    {
     Keypad_button_clicked(1, 3); //ASCII value of B
    }
    else if(GPIO_Pin == GPIO_PIN_0 && HAL_GPIO_ReadPin(GPIOD,
GPIO_PIN_0))
    {
     Keypad_button_clicked(0, 3); //ASCII value of A
    }
}
```

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі була проведена розробка системи, де спершу була розглянута загальна архітектура та обладнання, котрі будуть необхідні для даної роботи. Також, було описано процес розгортання програмного забезпечення, заразом із технологіями.

Був описаний принцип взаємодії із клавіатурою на платі STM32F4, загальний алгоритм роботи із нею, а також наведено адреси необхідних регістрів для програмування.

Було розглянуто можливі варіанти передачі даних, введених із клавіатури до ПК, вибір було здійснено на користь Virtual Com порту.

Було розроблено проект у середовищі STM32 CubeIDE, котрий складається із драйвера клавіатури. Система включає в себе не тільки розробку із безпосереднім поллінгом клавіш, але й додаткову модифікаційну версію із перериванням для оптимального використання ресурсів мікроконтролера та периферійних елементів.

РОЗДІЛ 4
ТЕСТУВАННЯ СИСТЕМИ

Беручи до уваги той факт, що в попередніх розділах був проведений огляд теоретичних матеріалів відносно теми роботи та проведена розробка програмного забезпечення (драйверу) для роботи із клавіатурою – в цьому розділі необхідно провести відлагодження програмного коду та протестувати роботу програми.

4.1 Компіляція та проєкт із SWV

Першим кроком після написання коду, коли головні компоненти проекту розглянуті – необхідно провести компіляцію STM32 створених проектів, як наведено на рисунку 4.1.

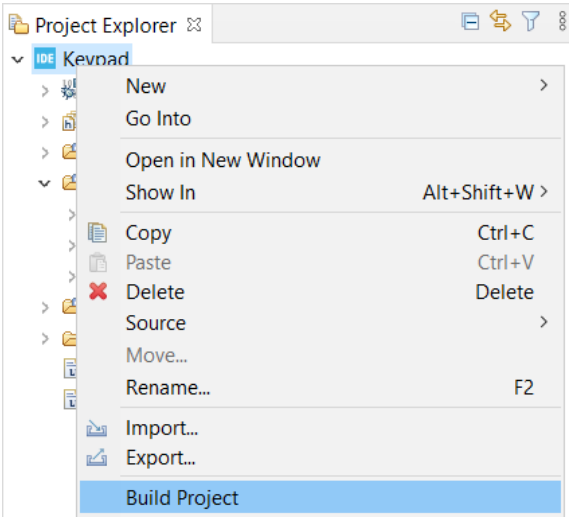


Рисунок. 4.1 – Компіляція STM32 проекту

При виникненні помилок після компіляції – спробуйте переглянути попередження та помилки, які виведені в консолі. Топ можливих причин, через які програма не може успішно скомпілюватися наведені в примітках даного розділу нижче.

Потім, необхідно провести налаштування конфігурації відлагодження проекту – а саме, натиснути Debug As -> Debug Configurations, я наведено на рисунку 4.2.

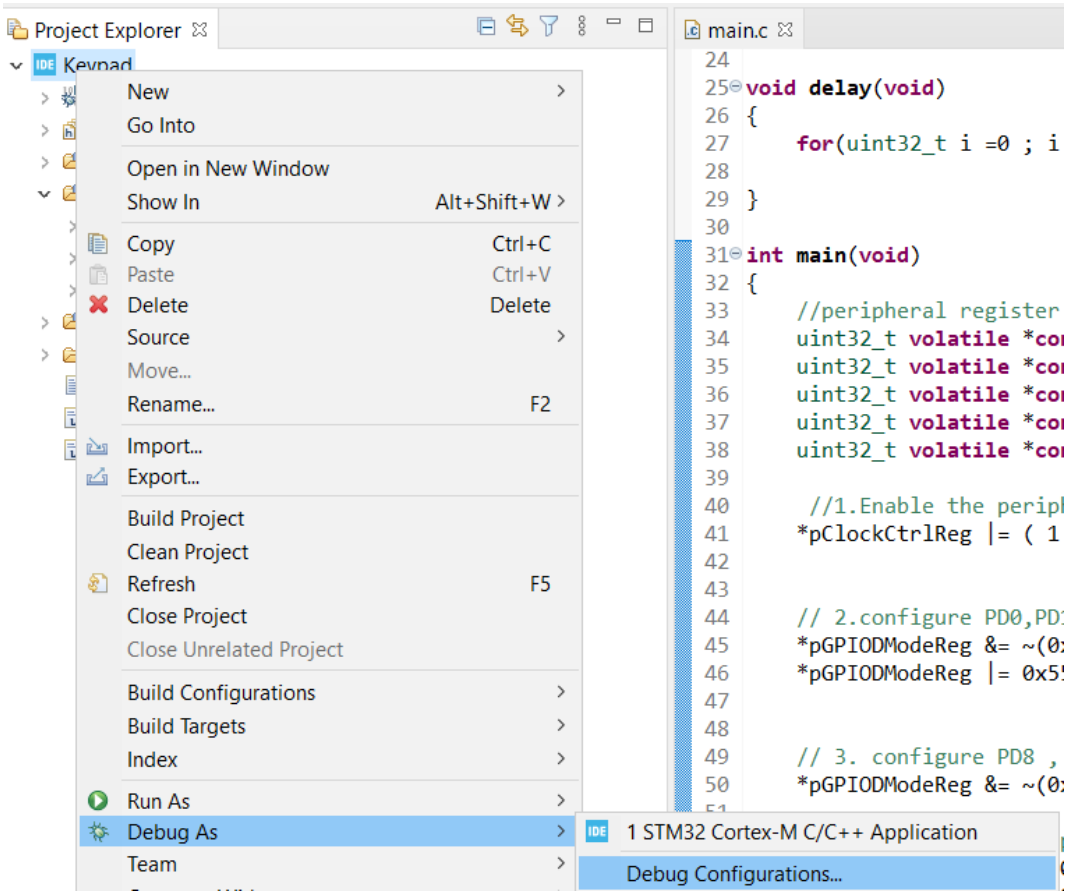


Рисунок 4.2 - Можливість налагодження STM32 проекту

В налаштуваннях – необхідно вказати для нашого проекту тип відлагодження – Debug probe – ST-Link GDB server, як наведено на рисунку 4.3. Даний інтерфейс для відлагодження зручно використовувати, оскільки даний тип уже вбудований всередину нашого мікроконтролера, проте, можна використати інший, зовнішній, за бажанням. Проте його налаштування займе додатковий відрізок часу.

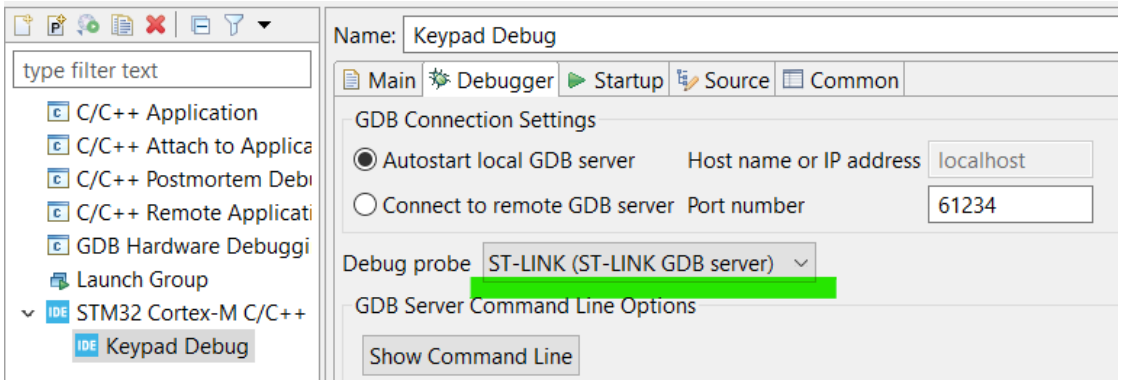


Рисунок 4.3 - Налаштування налагодження STM32 проекту

4.2 Налаштування SWV

Далі, налаштуємо SWV (serial wire viewer) консоль та скористаємося нею для перевірки виведення тексту ITM_. Для цього ввімкнемо SWV, натиснувши на checkbox в налаштуваннях проекту, як показано на рисунку 4.4.

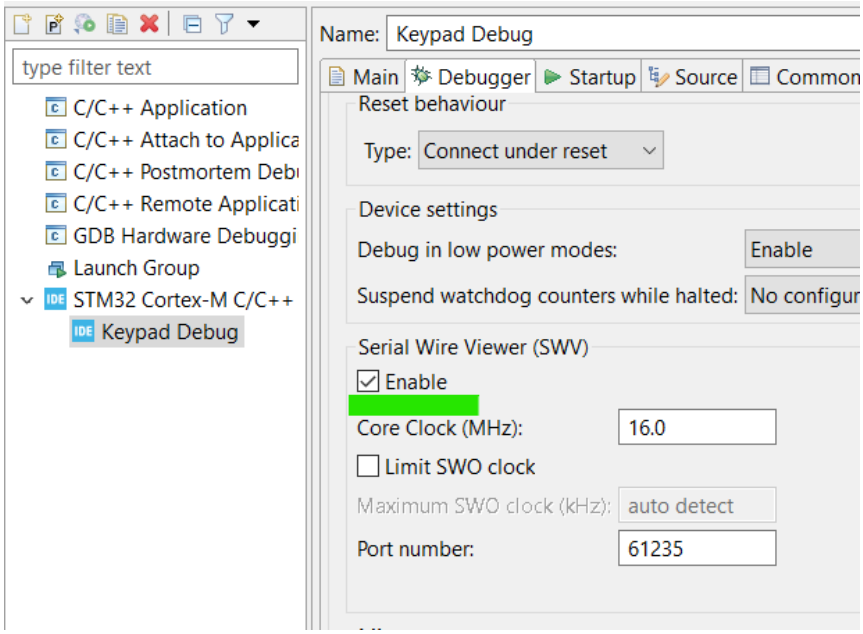


Рисунок 4.4 - Підключення SWV в STM32 проекті

Далі, потрібно безпосередньо вказати відкриття консолі в STM32 CubeIDE, а саме, як на рисунку 4.5, вибрати Window -> Show View -> SWV -> SWV ITM Data Console.

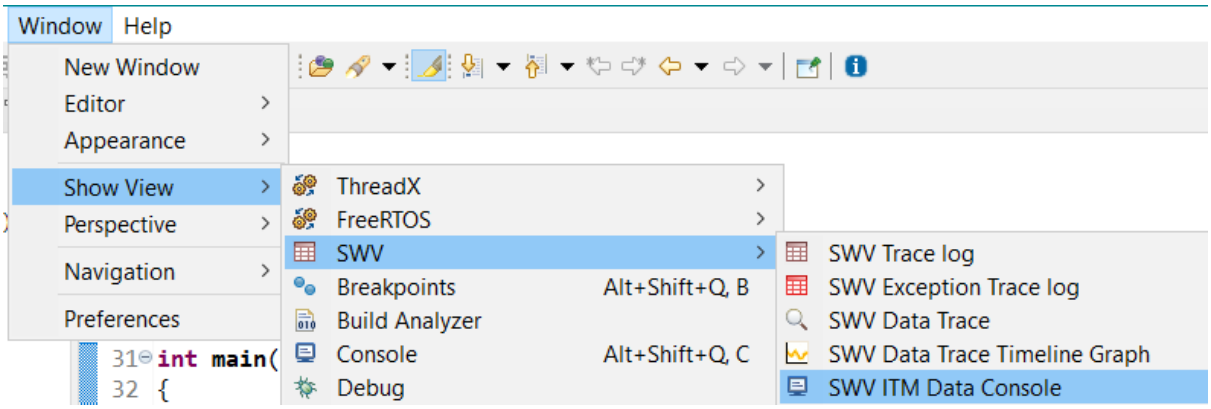


Рисунок 4.5 - Перехід до SWV консолі в STM32 проекті

Для налаштування ITM порту та відстежування повідомлень потрібно виконати наступні дії, котрі додатково вказані на рисунку 4.6 :

1) Натиснути кнопку - Configure trace
2) Поставити галочку на ITM Stimulus Ports – 0 порт

3) Натиснути кнопку Ок

4) Натиснути кнопку Start trace

5) В debug панелі натиснути Resume, щоб дійти до циклу while(1) написаної програми

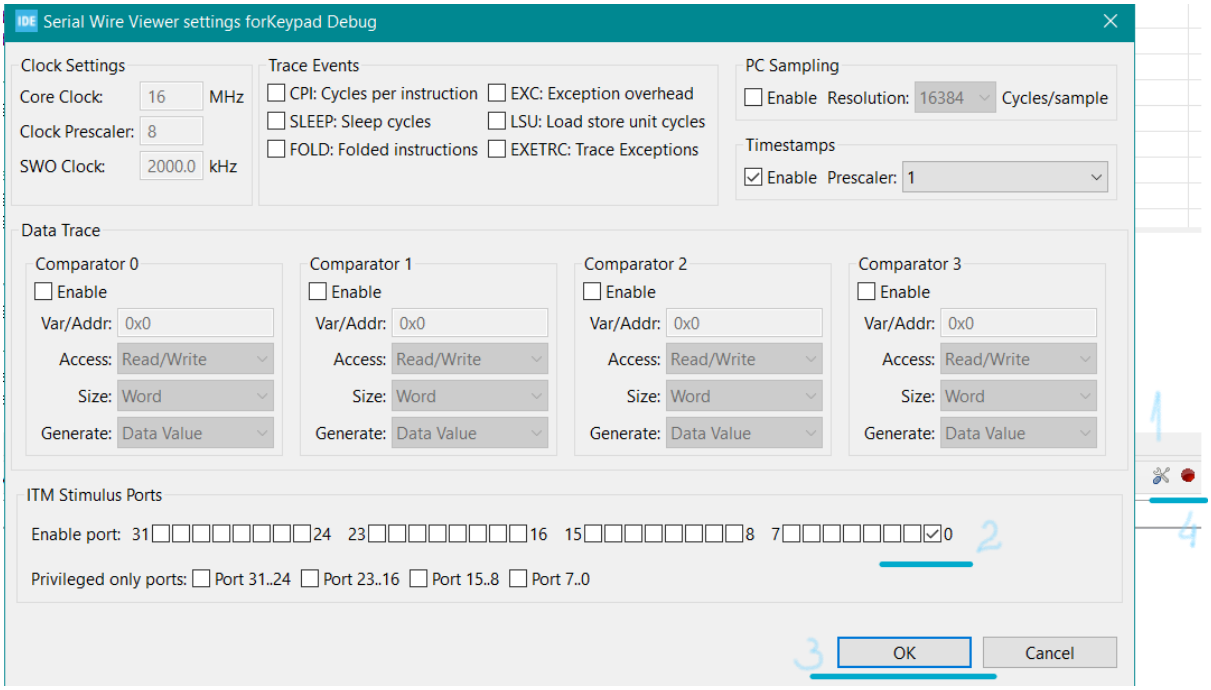


Рисунок 4.6 - Налаштування SWV в STM32 проекті

Наступним кроком є – ввімкнення кнопки Resume, яка в режимі дебаггінгу буде виконувати тіло цикла while, тобто, відстеження натиснутих кнопок на клавіатурі, якщо ми розглядаємо версію без переривань. Вказано необхідну кнопку на рисунку 4.7.

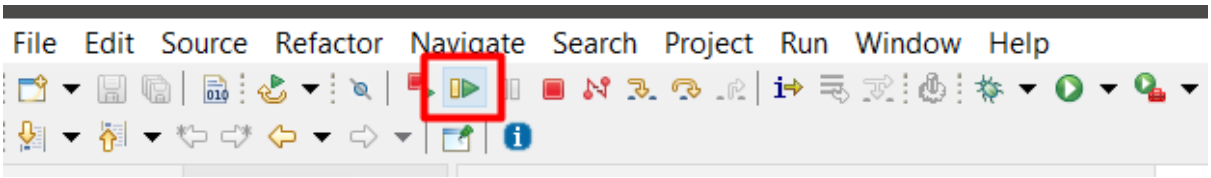


Рисунок 4.7 - Налаштування SWV в STM32 проекті

При умові успішного виконання попередніх алгоритмічних дій – повинна відкритися консоль. Тепер, натискаємо на кнопки клавіатури – і якщо все виконано вірно - у консолі з’являється виведення відповідно натиснутих символів, що означатиме правильне виконання написаної програми. Приклад введених мною символів можна побачити на рисунку 4.8.

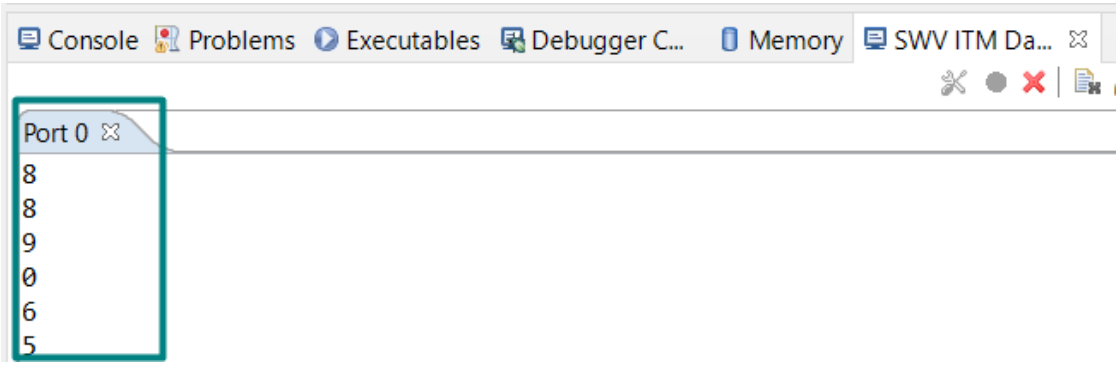


Рисунок 4.8 - Виконання програми в консолі SWV STM32

Тепер розглянемо наступний етап – удосконалену версію використання клавіатури із перериваннями та дисплеєм (там було використано драйвер для роботи із WH1602H). Додано в проект драйвер дисплею і тепер замість SWV консолі використано вивід результату перетворення числа із попереднього коду на дисплей. Завдання, по суті, зміниться тим, що потрібно буде викликати функцію Display_Write_Data(char) замість printf(char). Компіляція та відлагодження відбувається аналогічним чином – так, як наведено у цьому розділі вище. Тому – при успішному виконанні програми на Run режимі - введіть дані на клавіатурі – будь-які 4 символи та натисніть підтвердження – символ '*'. Відобразиться введений код в шістнадцятковій системі числення, оскільки функція цієї кнопки – конвертація із бінарної системи. Приклад введення можна спостерігати на рисунку 4.9. Повний лістинг програми наведено у додатку 5.

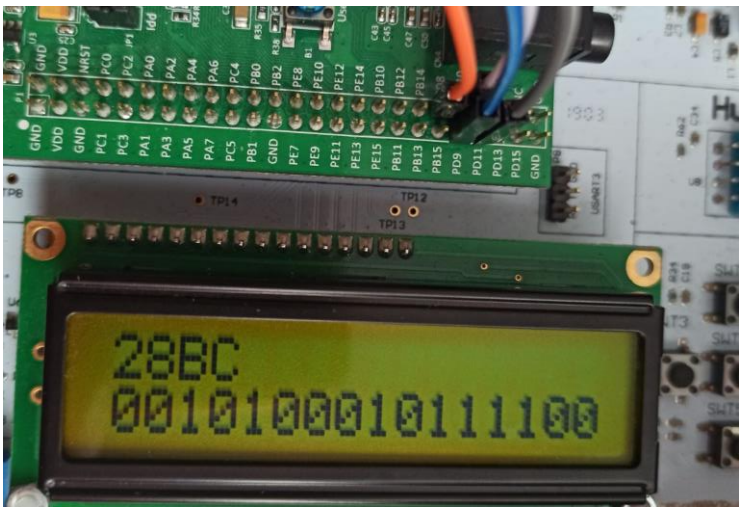


Рисунок 4.9 - Приклад виконання програми на дисплеї GL Starter Kit

Також, варто наголосити на топі можливих причин неробочої програми:

- Від’єдналися дроти-піни від клавіатури до плати.
- Деякі кнопки клавіатури можуть бути неробочими, оскільки клавіатура дешева.
- При сильному натисненні на кнопку ключ можу замкнутися і не розімкнутися, тому потрібно буде знизу натиснути на кнопку (тобто, у зворотньому напрямі).
- Якщо друкуються рандомні, хаотичні символи – не відбулася ініціалізація клавіатури, або невідповідність між підключеними дротами та налаштуванням у програмі.

4.3 Завантаження Virtual COM Port Driver для проекту

Також, варто додати можливість передачі даних із клавіатури плати STM32 до комп’ютера безпосередньо, використавши при цьому якийсь спосіб передачі даних, котрий займатиме невеликі затрати на його ініціалізацію та передаватиме дані з високою швидкістю. Як було розглянуто в попередньому розділі – для цього відмінно підійшов віртуальний COM порт.

Для початку – завантажимо STM32 Virtual COM Port Driver. Зараз, на офіційному веб-сайті (https://www.st.com/en/) його остання версія - STSW-STM32102 v1.5.0. Для завантаження потрібна повна реєстрація або часткова. Як на рисунку 4.10 нижче– вводимо дані та тиснемо на «Завантаження».

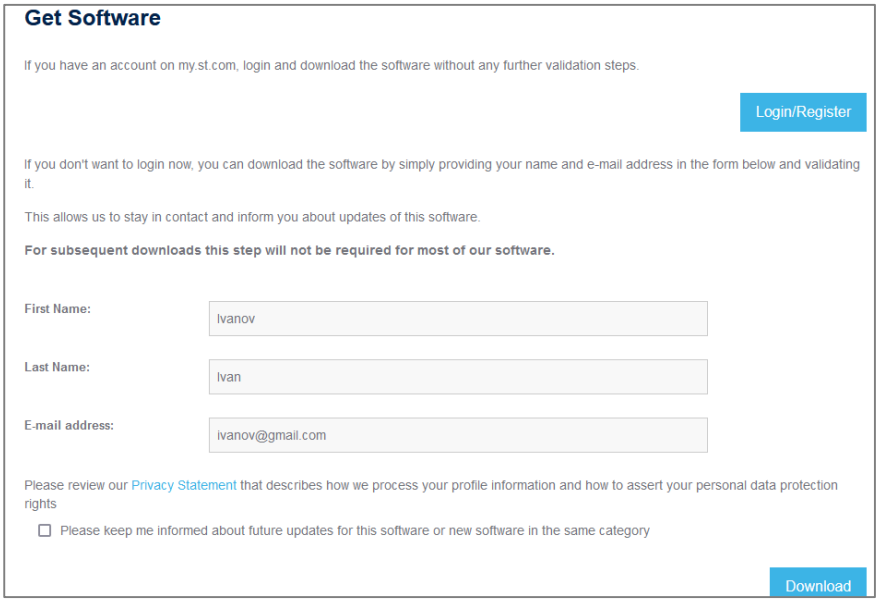


Рисунок 4.10 - Регістрація для STM32 Virtual COM Port Driver

На адресу вказаної електронної пошти – прийде лист, у якому буде надане посилання на Virtual COM Port.

Із отриманого листа, витягніть та встановіть отриманий .exe файл. Коли драйвер встановиться перейдіть до папки, вказаної на рисунку нижче та інсталюйте ще й цей застосунок.

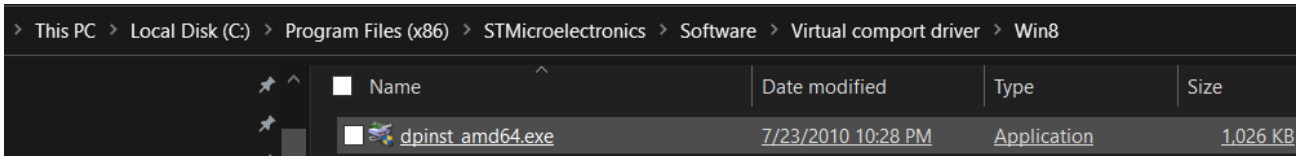


Рисунок 4.11 - Розміщення драйвера Virtual COM Port Driver

Драйвер встановлено, тому далі,

- Під'єднаємо плату до ПК через micro-USB для передачі даних по Virtual COM (не плутати з живленням по mini-USB).
- Перейдемо до Диспетчеру пристроїв та знайдемо у списку Порти, як вказано на рисунку 4.12.

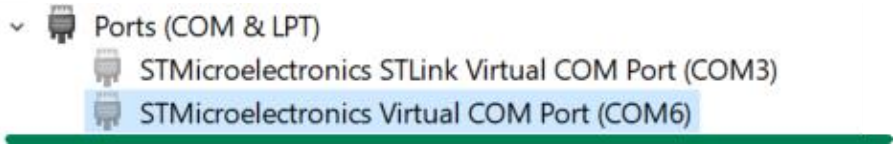


Рисунок 4.12 - STM32 Virtual COM Port Driver у Диспечері пристроїв

- В налаштуваннях STMicroelectronics Virtual COM Port налаштуємо шостий (6) порт, як показано на рисунку 4.13.

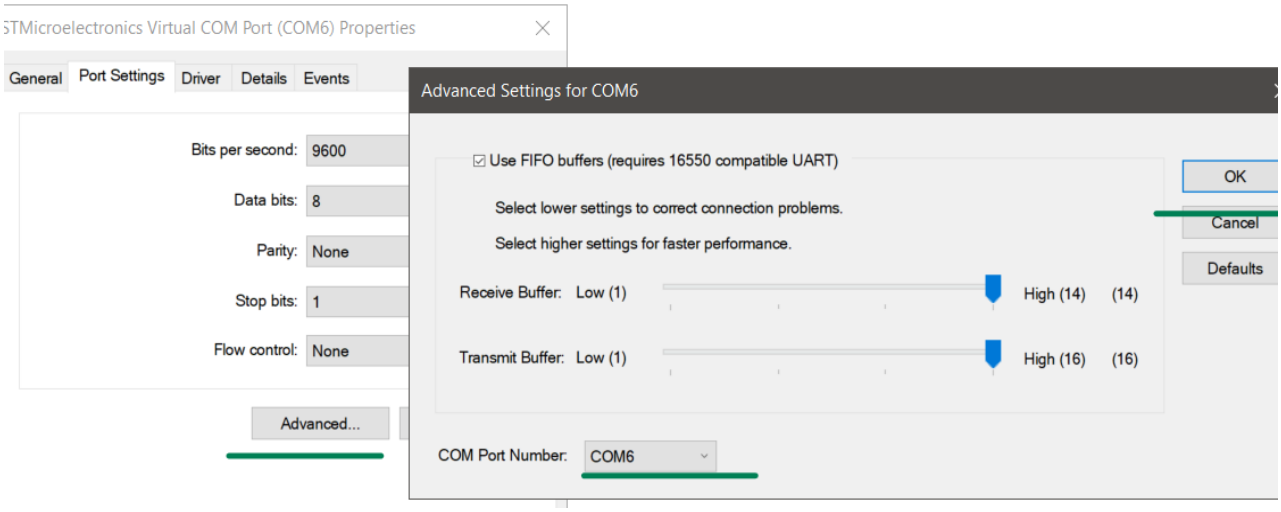


Рисунок 4.13 - Налаштування для STM32 Virtual COM Port Driver

4.4 Налаштування проекту із COM портом в STM32CubeIDE

Тепер можна перейти до створення нового STM32 проекту для STM32F407-Discovery в STMCubeIDE, де використаємо передачу даних за допомогою Virtual COM port 6.

Спершу, додамо I2C протокол послідовного зв'язку, котрий передає по лінійному дроті побітно по SDA. Він є синхронним, тому врегулювання правильної послідовності бітів регулюється тактовою частотою, котра спільна як для master, так і для slave, проте керує сигналом годинника завжди майстер. Налаштування наведено на рисунку 4.14.

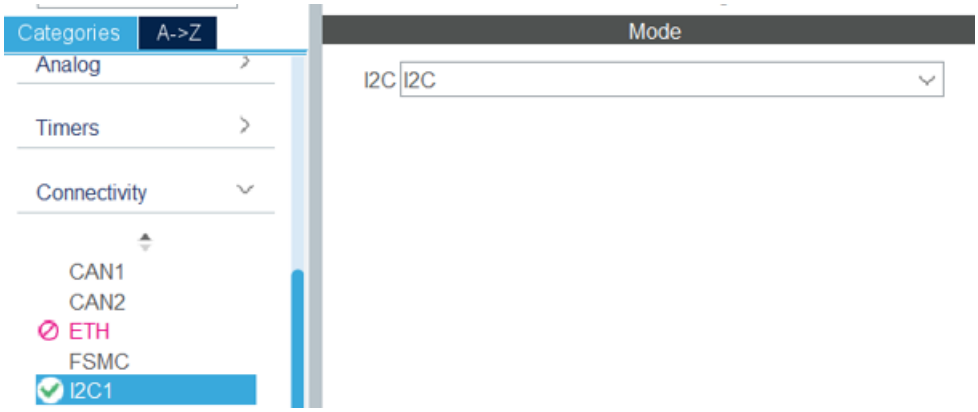


Рисунок 4.14 - Налаштування I2C в новому проєкті

Опісля, потрібно вказати вибір USB для зв'язку передачі – ми використаємо USB OTG Full-Speed в режимі Device-Only, пояснення до котрого було наведено в попередньому розділі. Деталі пропоную переглянути на рисунку 4.15.

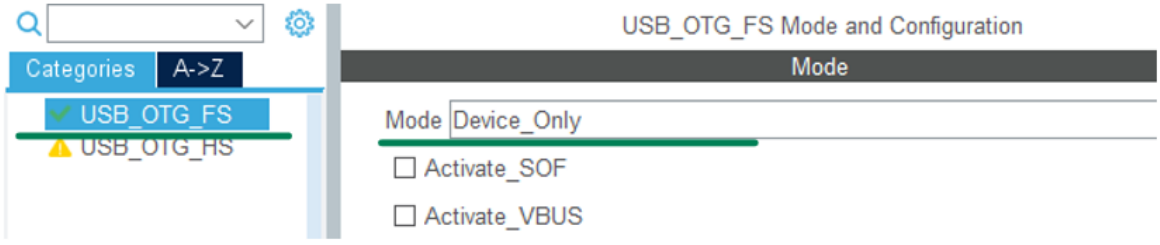


Рисунок 4.15 - Налаштування USB_OTG_FS у новому проєкті

Відповідно, необхідно додати налаштування типу зв'язку – передача по якому саме протоколу планується здійснювати. Тому, клас для FS IP вкажемо як Virtual Com port, як додано на рисунку 4.16.

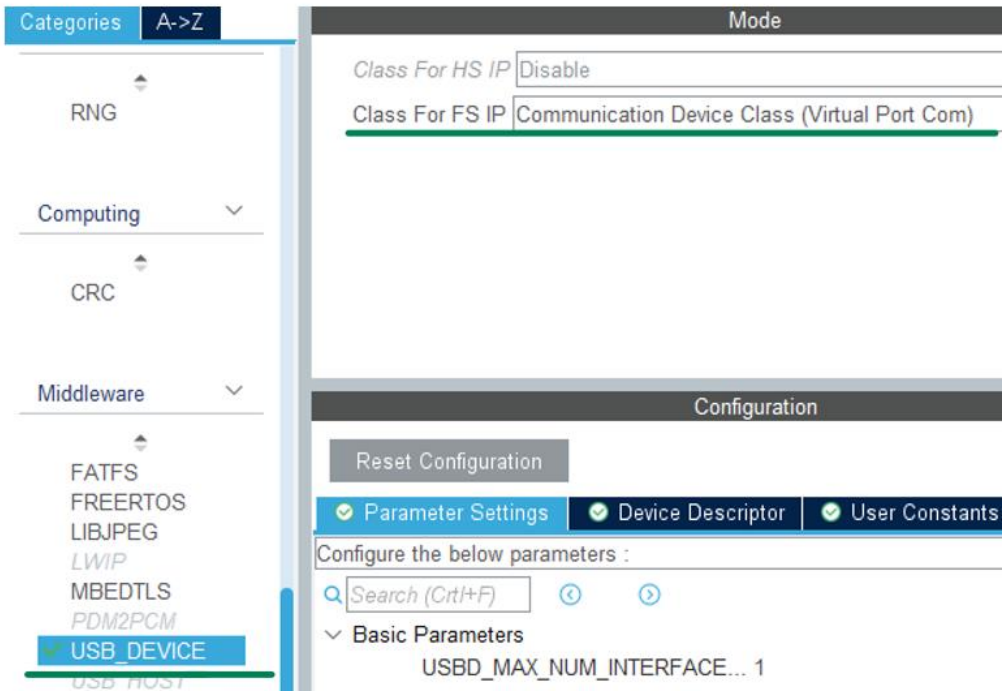


Рисунок 4.16 - Налаштування USB_HOST в FS у новому проєкті

Також, слід нагадати, що для USB використовуються глобальні переривання, саме тому для FS ми додамо його в таблиці NVIC переривань, як наведено на рисунку 4.17.

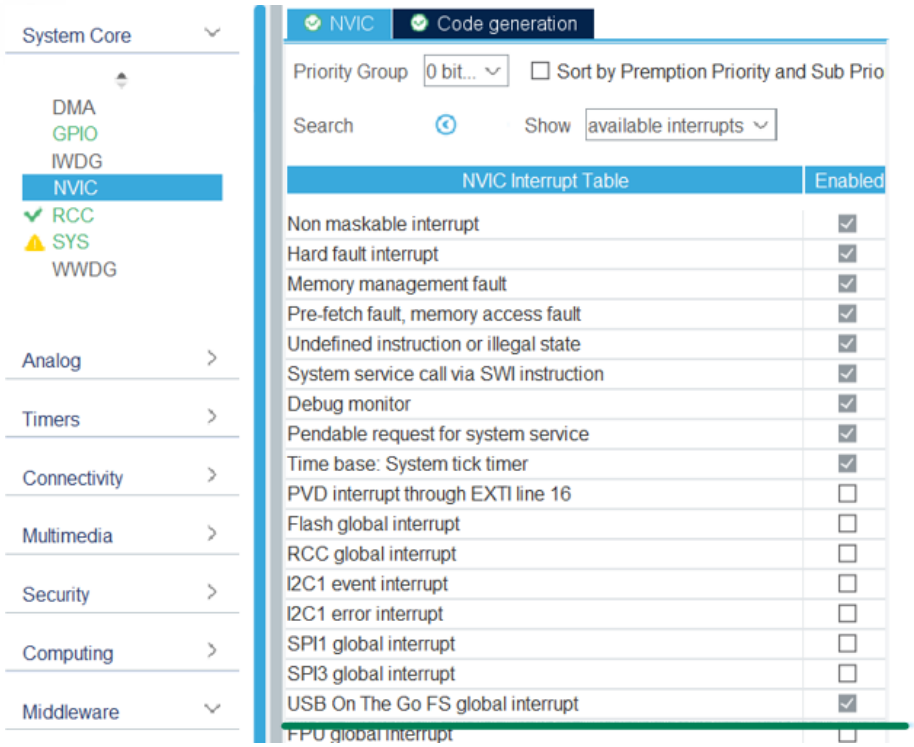


Рисунок 4.17 - Налаштування переривання від USB FS у новому проєкті

Якщо переглянути загальну характеристику частот на кожному елементі плати та периферії – то побачимо відповідні конфігурації годинників, як і зображено на рисунку 4.18. Частота HCLK тут залишилася максимально можливою для даного типу мікроконтролерів – 168 МГц.

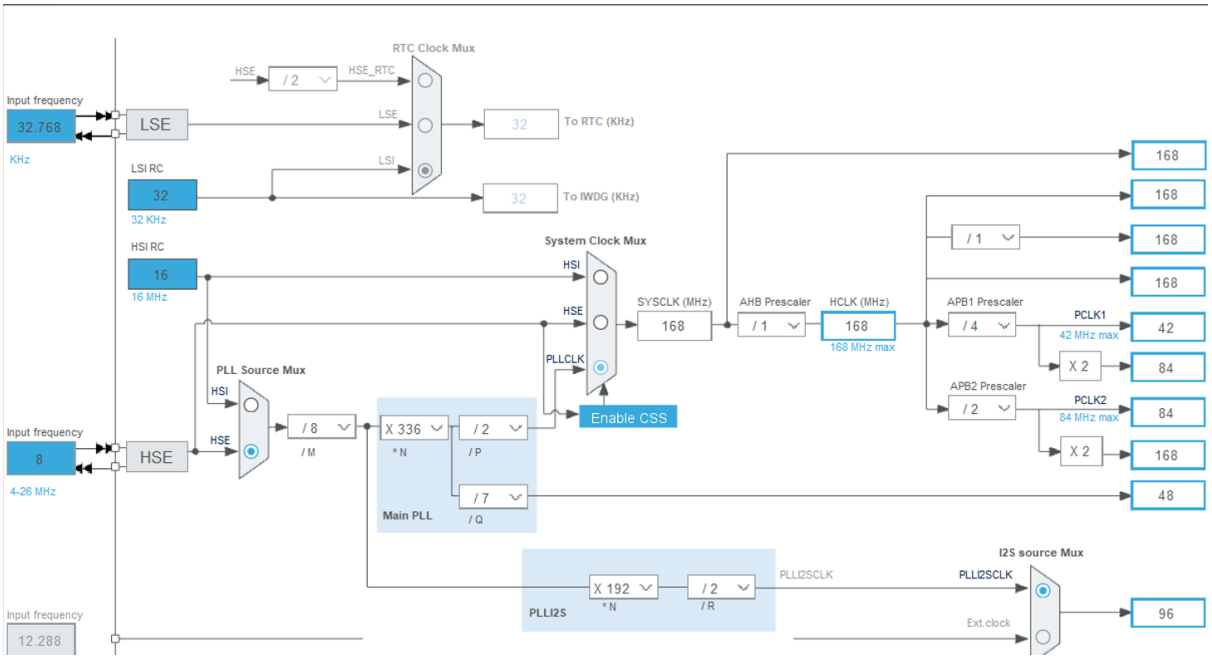


Рисунок 4.18 - Конфігурація Clock

Тепер, в нас є можливість успішної генерації проекту для STM32CubeIDE. Перейдемо до перегляд файлів із С програмним редагуванням – а саме, до файлу за шляхом - Middlewares => ST => STM32_USB_Device_Library => Class => CDC => Src => usbd_cdc.c.

Цей файл надає функції мікропрограмного забезпечення високого рівня для керування наступним функціоналом класу USB CDC:

- Ініціалізація та конфігурація високого та нижнього шарів програмної архітектури
- Перерахування (enums) як пристрій CDC (і перерахування для кожного реалізованого інтерфейсу пам'яті)
- Передача даних OUT/IN
- Передача команди IN (керування запитами класу)
- Керування помилками

Також, якщо переглянути опис драйвера класу CDC, бачимо наступне. Драйвер керує "Визначеннями класу універсальної послідовної шини для комунікації пристроїв". Він реалізує наступні аспекти специфікації:

- Керування дескрипторами пристрою
- Управління дескриптором конфігурації
- Перерахування (enums) як пристрій CDC з 2 кінцевими точками даних (IN і OUT) і 1 кінцевою точкою команди (IN)
- Керування запитами
- Функціональний збір Union (з використанням кінцевої точки 1 IN для контролю)

Функції, за допомогою яких будуть надсилатися та/або отримуватися дані (верхній шар) розміщені в **USB_DEVICE -> App -> usbd_cdc_if.c** файлі. Зокрема, функція **CDC_Transmit_FS (uint8_t* Buf, uint16_t Len)** використовується для передачі даних до ПК через USB інтерфейс. Її параметри - це **Buf** (буфер для надсилання) та **Len** (довжина даних).

У проєкті змінимо розмір буферу даних для надсилання та отримування даних відповідно до нашого COM6 порту, як зображено на рисунок 4.19.

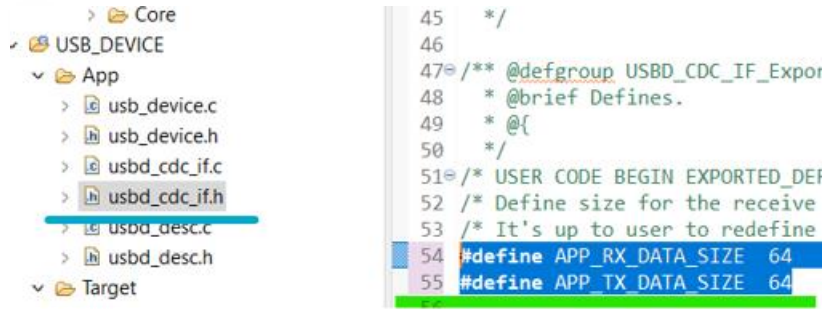


Рисунок 4.19 - Налаштування буферу передачі та отримування даних

Також, не варто забувати, що в нас є змога змінити максимально можливий пакет даних для передачі та отримування даних для HS USB – на 256U, як і вказано на рисунку 4.20 нижче.

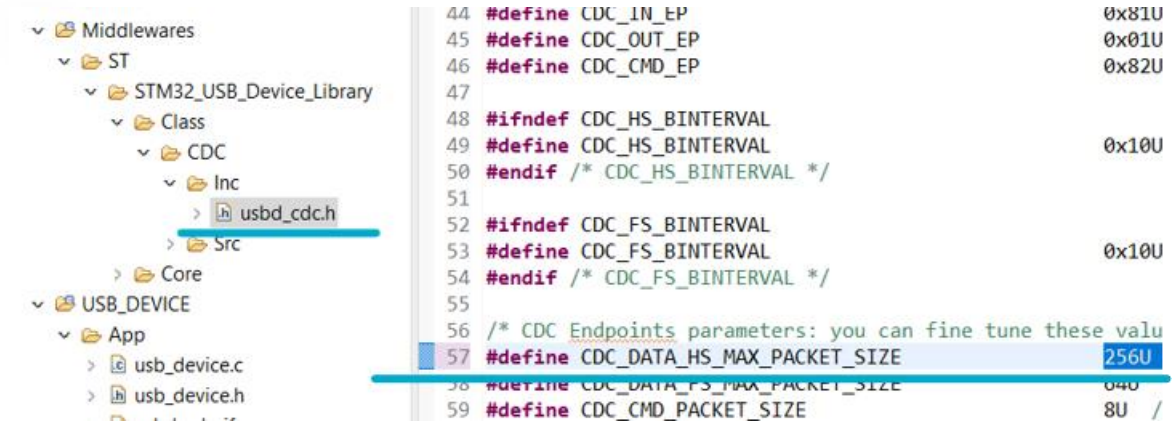


Рисунок 4.20 - Налаштування розміру пакету передачі/отримування даних

4.5 Перевірка COM порту

Аби перевірити, чи правильно виконана конфігурація проекту та присутня наявність передачі даних з плати до ПК – напишемо невелику програму, приклад коду якої наведений на рисунку 4.21.

```
uint8_t *data = "Hello World from USB CDC\r\n";
int main () {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USB_DEVICE_Init();
  while (1)
  {
      // transmits data via COM port to PC every 1 second
      CDC_Transmit_FS(data, strlen(data));
      HAL_Delay (1000);
  }
}
```

Рисунок 4.21 – Приклад перевірки COM порту

Також, для перегляду отриманих даних від плати до ПК - необхідно завантажити термінал Tera Term на ПК, котрий нарисований на рисунку 4.22.

Рисунок 4.22 – Програма-термінал Tera Term

Функція CDC_Transmit_FS перед тим, як надіслати пакет з буферу перевіряє стан пристрою в режимі надсилання даних - TxState. Якщо Tx_State != 0 => USBD_BUSY, тобто USB Device зайнятий. Якщо ні, то за замовчуванням буде стан USBD_OK, тобто вільний.

При стані пристрою USBD_OK у буфер пристрою записується дані із вхідного параметру функції – Buf , дивитися на рисунку 4.23, 291 рядок). Опісля, пристрій надсилає пакет із буфера до ПК.

```
283 uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
284 {
285   uint8_t result = USBD_OK;
286   /* USER CODE BEGIN 7 */
287   USBD_CDC_HandleTypeDef *hcdc = (USBD_CDC_HandleTypeDef*)hUsbDeviceFS.pClassData;
288   if (hcdc->TxState != 0){
289     return USBD_BUSY;
290   }
291   USBD_CDC_SetTxBuffer(&hUsbDeviceFS, Buf, Len);
292   result = USBD_CDC_TransmitPacket(&hUsbDeviceFS);
293   /* USER CODE END 7 */
294   return result;
295 }
```

Рисунок 4.23 - Імплементація функції CDC_Transmit_FS

Після Компіляції(Build) та Запуску(Run) проєкту – у Диспетчері пристроїв бачимо новий пристрій, під’єднаний по USB (Virtual COM Port) до ПК. Налаштуємо термінал на serial COM6 port таким чином, як показано на рисунку 4.24.

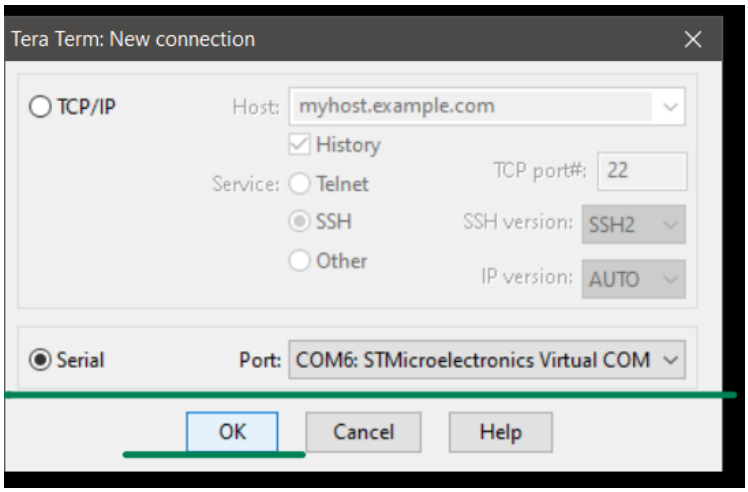


Рисунок 4.24 - Налаштування терміналу Tera Term

Після проведених дій бачимо скріншот із терміналу (COM6) про успішну передачу даних. Тому – робимо висновок про успішну перевірку.

Зображення скріншоту із введеними символами бачимо на рисунку 4.25.

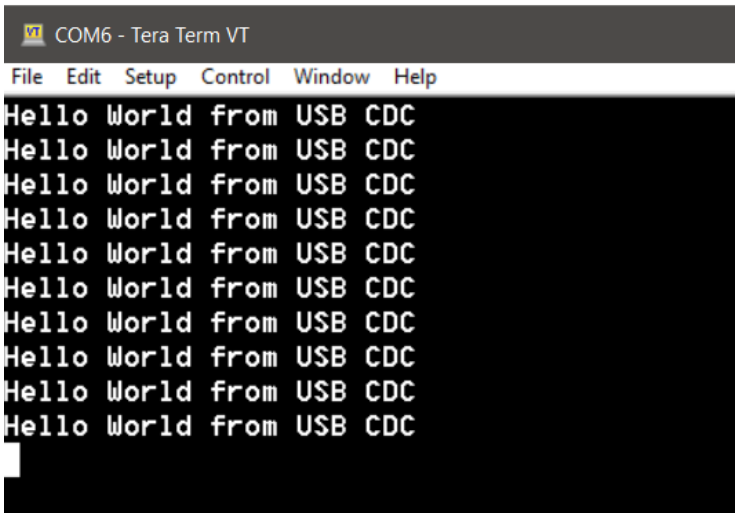


Рисунок 4.25 - Результат виконання програми із терміналу Tera Term

Тепер відредагуємо поточний проект, додавши драйвер роботи із клавіатурою 4x4 – файли keypad.h, keypad.c та main функцію із коду, де відсутні переривання.

Після Компіляції(Build) та Запуску(Run) проєкту – у Диспетчері пристроїв бачимо новий пристрій, під'єднаний по USB (Virtual COM Port) до ПК. Також, знову налаштуємо термінал на serial COM6 port.

При натисканні клавіш відбувається передача символів. Зображення із терміналу (COM6) про успішну передачу даних наведено на рисунку 4.26. Перевірка пройдена успішно!

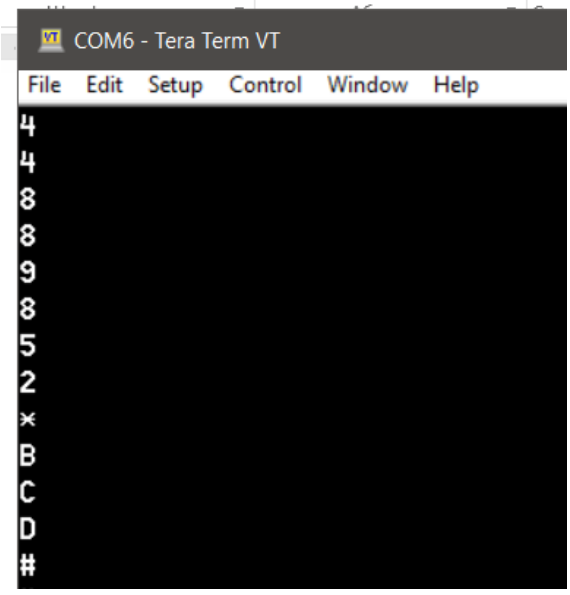


Рисунок 4.26 - Результат виконання програми із терміналу Tera Term

Також, наведемо певні примітки до написаного коду.

1. Для накопичення натиснутих символів можна використати масив типу char або uint8_t (залежно від початкової системи числення).
2. Для перетворення числа із однієї системи числення в іншу – створити окрему функцію, наприклад hex_to_dec();
3. Додатково, можете під’єднати драйвер для дисплею та виводити вхідні та вихідні дані ще й туди. Це зручно й для дебагінгу, бо так зможете переконатися, що дані із клавіатури надходять до плати. Тобто, якщо не буде відбуватися передача через COM порт – проблема могла виникнути із драйвером або вхідні дані для передачі через функцію CDC_Transmit_FS сформовані неправильно.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі було розглянуто компіляцію та налаштування проекту в програмному середовищі STM32 CubeIDE. Окрім того, було проведено відлагодження програмного забезпечення для драйвера клавіатури 4x4 через вбудований ST-Link Debugger Probe до мікроконтролера.

Було показано проєкт із консоллю SWV, яка була використана для відображення символів, введених із клавіатури, як найпростіший спосіб перевірки.

Було створено додатковий проєкт для перевірки передачі даних до ПК – через віртуальний COM порт та протестовано отримання бітів за допомогою термінату Tera Term.

Було продемонстровано відлагодження готового проектного рішення та запропоновано топ можливих причин через які неможливе успішне виконання програми чи можливі апаратні проблеми.

ВИСНОВОК

В результаті виконання дипломного проєкту було реалізовано навчально-апаратний комплекс для роботи із клавіатурою 4x4 для STM32.

У першому розділі було проведено роботу щодо аналізу актуальності та доцільності розробки даного проекту та проагалізовано ситуацію на ринку сучасних курсів та детально розглянуто опис кожного вибраного курсу по актуальній темі.

У другому розділі була проведена проєктна розробка над оглядом технологій, котрі необхідні для створення проекту та загальну архітектурну структуру. Спершу, було проаналізовано 32-бітні плати. Відбувся детальний розгляд переваг та недоліків кожного варіанту, отриманих із документацій та таблиць із характеристиками та було обрано STM32F407DISCOVERY мікроконтролер.

У третьому розділі було описано процес розгортання програмного забезпечення, заразом із технологіями. Також, був описаний алгоритм взаємодії із клавіатурою на платі STM32F4, а також наведено адреси необхідних регістрів для програмування. Було розроблено проект у середовищі CubeIDE, котрий складається із драйвера клавіатури. Система окрім розробки програми із безпосереднім поллінгом клавіш, містить ще й додаткову модифікаційну версію із перериванням для ефективного використання ресурсів плати й периферійних елементів.

У четвертому розділі було розглянуто компіляцію та налаштування проєкту в програмному середовищі та проведено відлагодження програмного забезпечення для драйвера клавіатури 4x4 через вбудований ST-Link Debugger мікроконтролера. Було показано проєктне рішення із консоллю SWV та перевірено працездатність через віртуальний COM порт. Було продемонстровано відлагодження проекту та наведено топ можливих проблем.

Під час виконання проєкту були вдосконалені теоретичні та практичні навички із роботою над embedded технологіями.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Embedded Systems Tutorial: What is, History & Characteristics [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/embedded-systems-tutorial.html
2. Mastering Microcontroller and Embedded Driver Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/mastering-microcontroller-with-peripheral-driver-development
3. Embedded Systems Programming on ARM Cortex-M3/M4 Processor | Udemy [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/embedded-system-programming-on-arm-cortex-m3m4/
4. Complete ARM Cortex-M Bare-Metal Programming (TM4C123) | Udemy [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/cortex-m-internals-master-pointers-structures-memory-etc/
5. IoT Application Development with the ESP32 Using the ESP-IDF | Udemy [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/iot-application-development-with-the-esp32-using-the-esp-idf/
6. What is an Arduino? - learn.sparkfun.com [Електронний ресурс] – Режим доступу до ресурсу: https://learn.sparkfun.com/tutorials/what-is-an-arduino/all
7. ESP32 vs ESP8226 Pros and Cons [Електронний ресурс] – Режим доступу до ресурсу: https://makeradvisor.com/esp32-vs-esp8266/
8. What is RaspBerry PI [Електронний ресурс] – Режим доступу до ресурсу: https://opensource.com/resources/raspberry-pi
9. STMicroelectronics STM32F4DISCOVERY | nanoFramework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.nanoframework.net/content/community-targets/stm32f4-discovery.html

10. STM32F4 - ARM Cortex-M4 High-Performance MCUs - STMicroelectronics [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html#featured_resources-0
11. Pros & cons of using STM32CubeMX code generation tool insead of manually writing drivers for an ARM Cortex-M microcontroller - Data Respons [Електронний ресурс] – Режим доступу до ресурсу: https://datarespons.com/pros-cons-using-stm32cubemx-code-generation-tool-insead-manually-writing-drivers-arm-cortex-m-microcontroller/
12. DataSheet для плати STM32F4 [Електронний ресурс] – Режим доступу до ресурсу: STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs - Reference manual-
13. Interrupt Handling – an overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/engineering/interrupt-handling#:~:text=Interrupt%20handling%20is%20a%20key,handler%20in%20the%20interrupt%20table
14. USB OTG HS [Електронний ресурс] – Режим доступу до ресурсу: https://community.st.com/s/question/0D50X00009XkgMSSAZ/stm32f4discovery-hs-usb
15. STM32 USB-Device vs USB OTG HS [Електронний ресурс] – Режим доступу до ресурсу: https://qastack.ru/electronics/234516/stm32-usb-device-vs-usb-otg-hs-what-is-the-difference
16. STM32: контролер USB OTG full-speed [Електронний ресурс] – Режим доступу до ресурсу: http://microsin.net/programming/arm-working-with-usb/stm32-usb-otg-full-speed.html
17. STM32: контролер USB OTG high-speed [Електронний ресурс] – Режим доступу до ресурсу: http://microsin.net/programming/arm-working-with-usb/stm32-usb-otg-high-speed.html
18. Serial Wire Viewer [Електронний ресурс] – Режим доступу до ресурсу: https://www.codeinsideout.com/blog/stm32/swv/

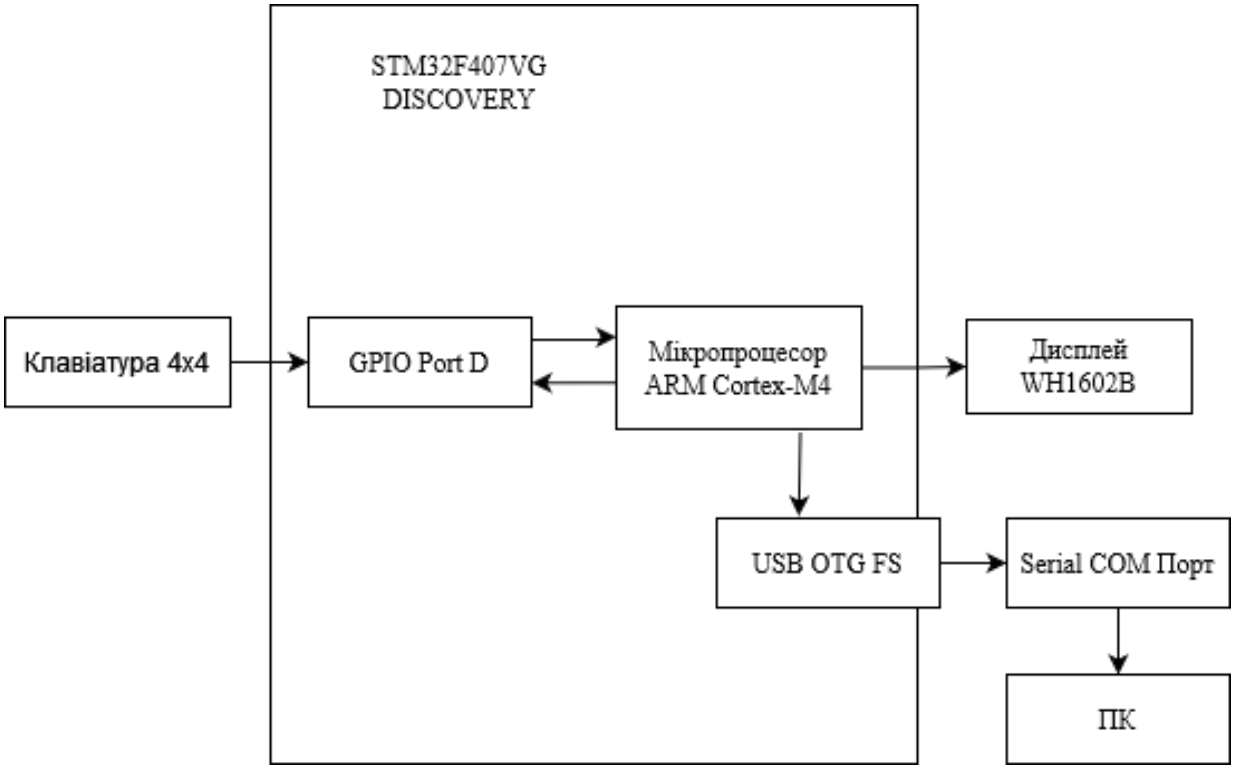
ДОДАТОК 1

Структурна схема

ІАЛЦ.467200.004 Д1

Листів 1

2022



					ІАЛЦ.467200.004 Д1
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розробив		*Яшан О.В.*			*Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32* *Структурна схема*
Перевір.		*Таранюк В.А.*			
Н. контр.		*Сімоненко В.П.*			
Затверд.		*Стіренко С.Г.*			

Літ.	*Аркуш*	*Аркушів*
	1	1

КПІ ім. Ігоря Сікорського
ФІОТ ІВ-81

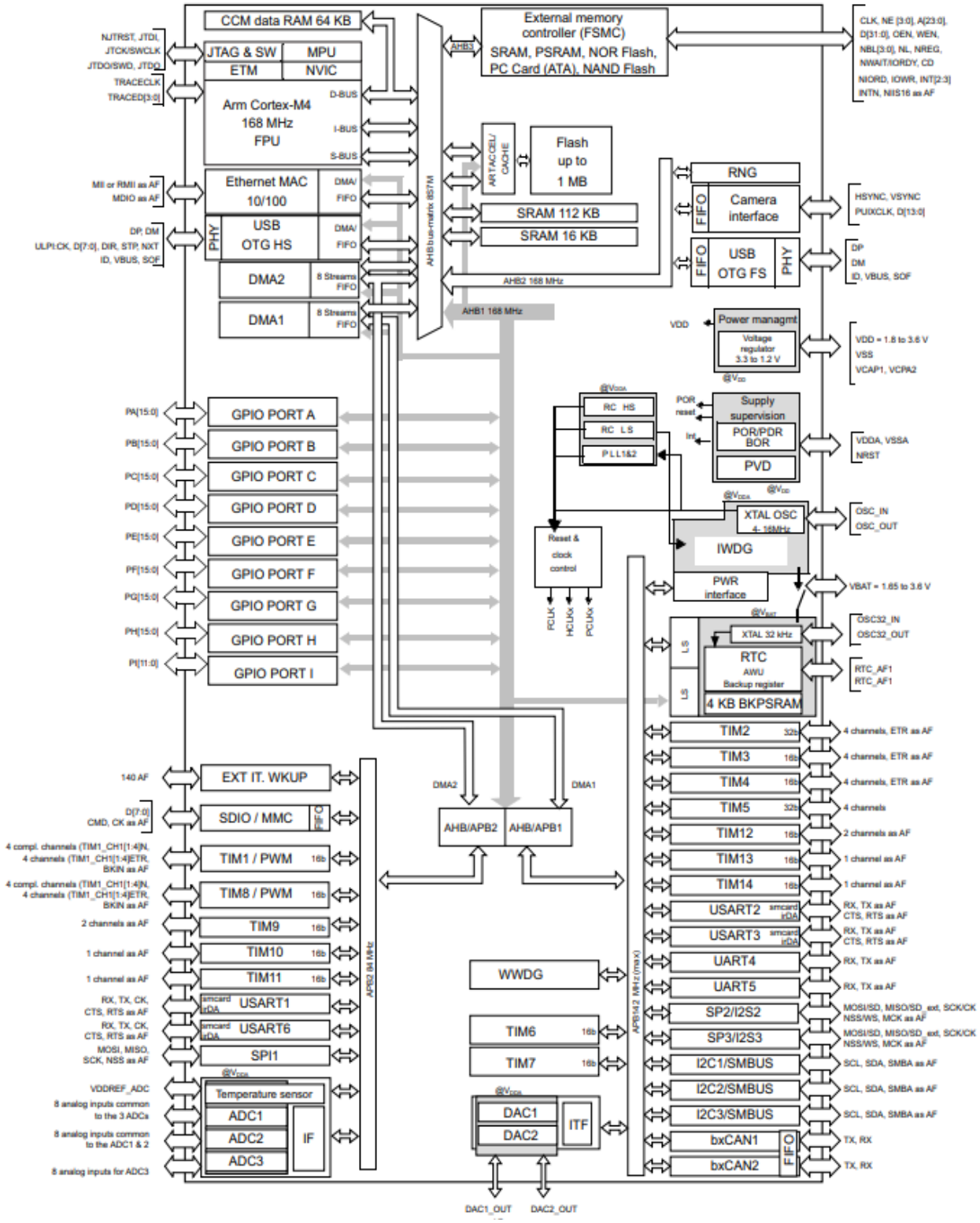
ДОДАТОК 2

Функціональна схема

ІАЛЦ.467200.005 Д2

Листів 1

2022



					ІАЛЦ.467200.005 Д2
Змін.	Арк.	№ докум.	Підпис	Дата	
Розробив		Яшан О.В.			Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32 Функціональна схема
Перевір.		Таранюк В.А.			
Н. контр.		Сімоненко В.П.			
Затверд.		Стіренко С.Г.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського
ФІОТ ІВ-81

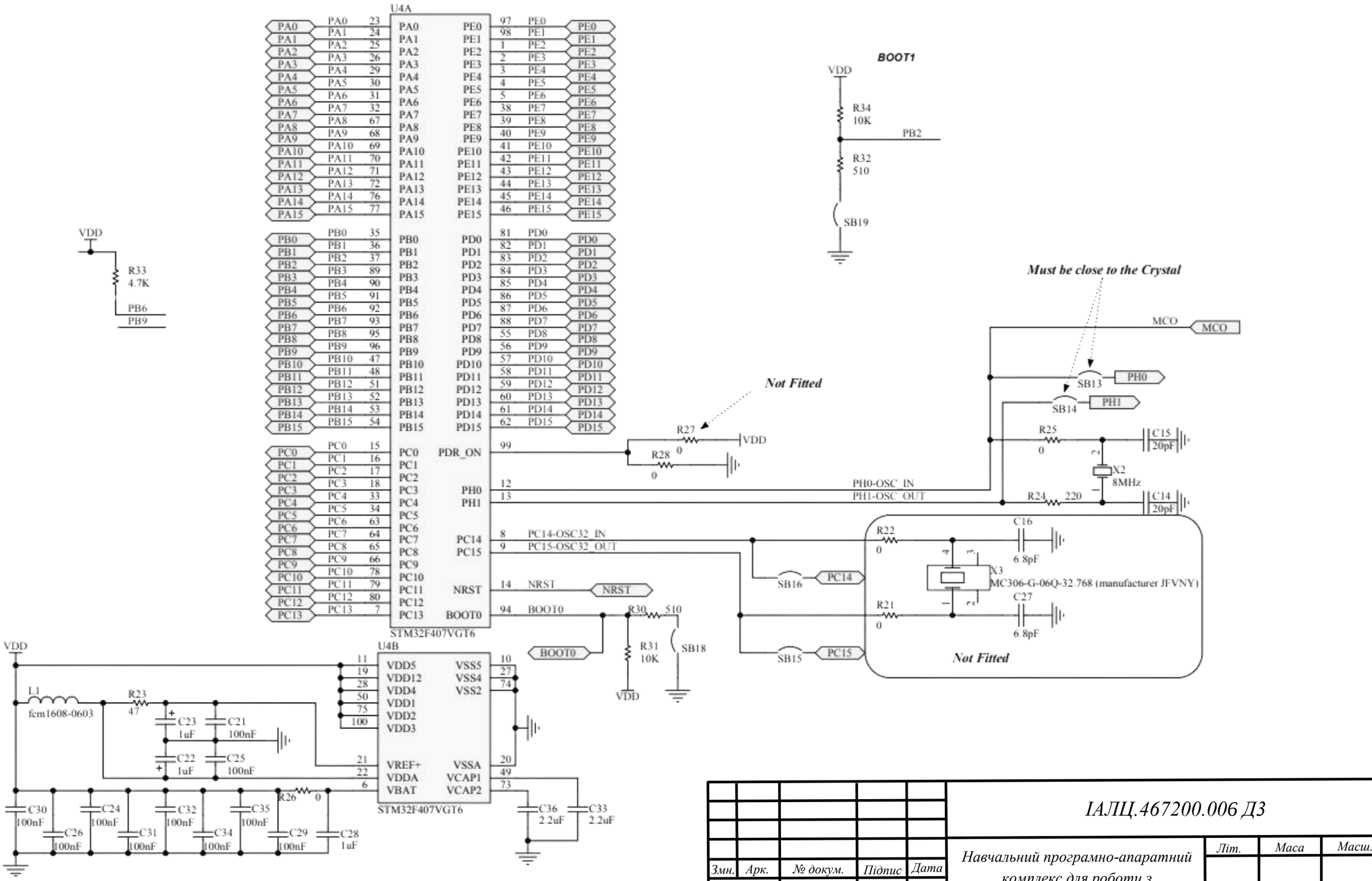
ДОДАТОК З

Принципова схема

ІАЛЦ.467200.006 Д3

Листів 1

2022



					ІАЛЦ.467200.006 Д3	
Змн.	Арк.	№ докум.	Підпис	Дата	Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32 Принципова схема	Літ. / Маса / Масш.
Розроб.		Яшан О.В.				
Перевір.		Таранюк В.А.				Аркуш 1 / Аркушів 1
Т. контр.					Дипломний проєкт	КПІ ім. Ігоря Сікорського ФІОТ ІВ-81
Н. контр.		Сімоненко В.П.				
Затверд.		Стіренко С.Г.				

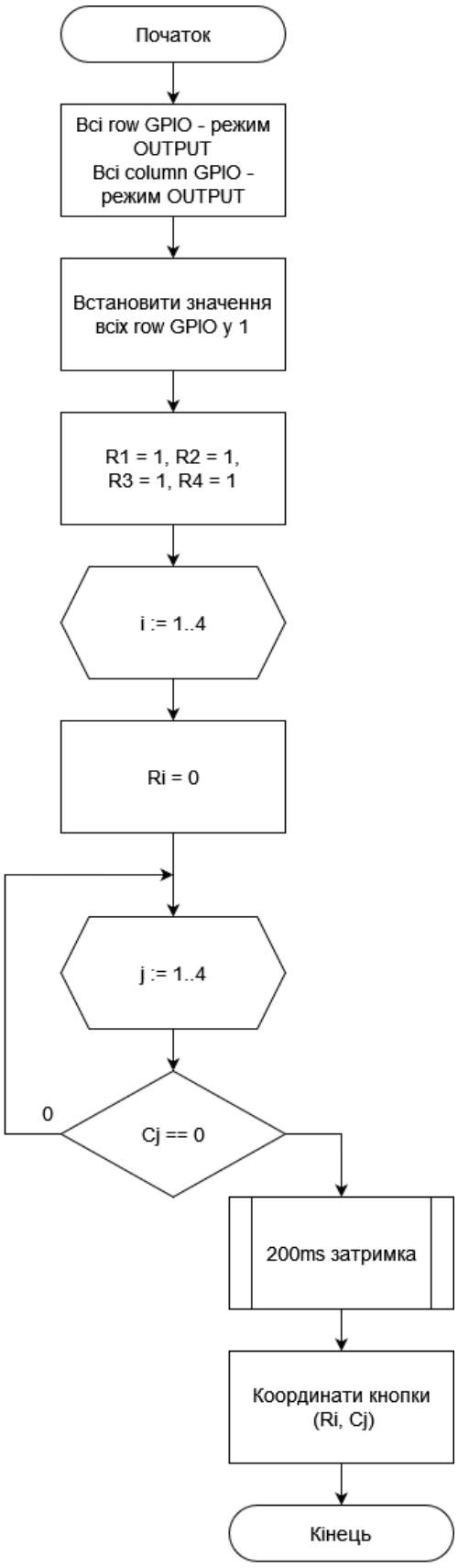
ДОДАТОК 4

Алгоритм імплементації

ІАЛЦ.467200.007 Д4

Листів 1

2022



					ІАЛЦ.467200.007 Д4	
Змін.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Яшан О.В.*			*Навчальний програмно-апаратний комплекс для роботи з клавіатурою в STM32 Алгоритм імплементації*	*Літ.* / *Аркуш* 1 / *Аркушів* 1
Перевір.		*Таранюк В.А.*				
Н. контр.		*Сімоненко В.П.*				КПІ ім. Ігоря Сікорського ФІОТ ІВ-81
Затверд.		*Стіренко С.Г.*				

ДОДАТОК 5

Текст програми

ІАЛЦ.467200.008 Д5

Листів 19

2022

main_simple.c

```
#include<stdint.h>
#include<stdio.h>

int main(void)
{
  //mendatory for keypad initialization – regs configuration
  Setup_GPIO_for_keypad();

  while(1)
  {
            Keypad_loop();
  }
}
```

keypad.h

```
#ifndef KEYPAD_H
#define KEYPAD_H

#include <string.h>
#include <stdio.h>
#include <stdint.h>
//peripheral register addresses
uint32_t volatile *const pGPIODModeReg  =  (uint32_t*)(0x40020C00);
uint32_t volatile *const pInPutDataReg  =  (uint32_t*)(0x40020C00+0x10);
uint32_t volatile *const pOutPutDataReg =  (uint32_t*)(0x40020C00+0x14);
uint32_t volatile *const pClockCtrlReg  =  (uint32_t*)(0x40023800+0x30);
uint32_t volatile *const pPullupDownReg =  (uint32_t*)(0x40020C00 + 0x0C);

const char* const keypad_symbols[16] = {"1\n", "2\n", "3\n", "A\n", "4\n", "5\n", "6\n", "B\n",
"7\n", "8\n", "9\n", "C\n", "*\n", "0\n", "#\n", "D\n"};

void Delay(void);
void Setup_GPIO_for_keypad(void);
void Convert_ (void);
void Keypad_loop(void);
#endif
```

keypad.c

```
#include "keypad.h"
void Delay(void)
{
        for(uint32_t i=0 ; i < 300000 ; i++);
}

void Setup_GPIO_for_keypad() {
          //1.Enable the peripheral clock of GPIOD peripheral
              *pClockCtrlReg |= ( 1 << 3);

          // 2.configure PD0,PD1,PD2,PD3 as output (rows)
```

```
		*pGPIODModeReg &= ~(0xFF); //clear
		*pGPIODModeReg |= 0x55;   //set

		// 3. configure PD8 , PD9, PD10, PD11 as input (columns)
	*pGPIODModeReg &= ~(0xFF << 16);

		// 4.Enable internal pull-up resistors for PD8 PD9 PD10 PD11
	*pPullupDownReg &= ~(0xFF << 16);
	*pPullupDownReg |=  (0x55 << 16);
}

void Convert_ () {
		// your implementation
}

void Keypad_loop() {

			for (uint8_t i=0; i<4; ++i) {

				//make all rows HIGH
				*pOutPutDataReg |= 0x0f;
				//make Ri LOW(PD0)
				*pOutPutDataReg &= ~( 1 << i);

				 //scan the columns
				//check C1 of Ri is low or high
				if(!(*pInPutDataReg & ( 1 << 8))) {
					//key is pressed

					//delay for only 1 char printed
					Delay();
					printf("%s", keypad_symbols[i*4+0]);
				}
				//check C2 of Ri is low or high
				if(!(*pInPutDataReg & ( 1 << 9))) {
					delay();
					printf("%s", keypad_symbols[i*4+1]);

				}
				//check C3 of Ri is low or high
				if(!(*pInPutDataReg & ( 1 << 10))) {
					Delay();
					printf("%s", keypad_symbols[i*4+2]);
				}
				//check C4 of Ri is low or high
				if(!(*pInPutDataReg & ( 1 << 11))) {
					Delay();
					printf("%s", keypad_symbols[i*4+3]);
				}
				Delay();
			}
}
```

syscalls.c

```
//Debug Exception and Monitor Control Register base address
#define DEMCR                   *((volatile uint32_t*) 0xE000EDFCU )

/* ITM register addresses */
#define ITM_STIMULUS_PORT0      *((volatile uint32_t*) 0xE0000000 )
#define ITM_TRACE_EN            *((volatile uint32_t*) 0xE0000E00 )

void ITM_SendChar(uint8_t ch)
{

        //Enable TRCENA
        DEMCR |= ( 1 << 24);

        //enable stimulus port 0
        ITM_TRACE_EN |= ( 1 << 0);

        // read FIFO status in bit [0]:
        while(!(ITM_STIMULUS_PORT0 & 1));

        //Write to ITM stimulus port0
        ITM_STIMULUS_PORT0 = ch;
}
```

main_interruptions.c

```
#include "main.h"
#include "gpio.h"
#include "drivers/DISPLAY_WH1602B_4Bit.h"

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
uint32_t previousMillis = 0;
uint32_t currentMillis = 0;


uint8_t data[16*2];
char keypad_symbols[16] = {'1', '2', '3', 'A', '4', '5', '6', 'B', '7', '8', '9', 'C', '*', '0', '#', 'D'};
uint8_t n_chars = 0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);

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

void Convert_hex_to_bin() {
                long int count=0;
```

```
                while(data[count])
                {
                        switch(data[count])
                        {
                                case '0' : Print_display_array("0000");
                                        break;
                                case '1' : Print_display_array("0001");
                                        break;
                                case '2' : Print_display_array("0010");
                                        break;
                                case '3' : Print_display_array("0011");
                                        break;
                                case '4' : Print_display_array("0100");
                                        break;
                                case '5' : Print_display_array("0101");
                                        break;
                                case '6' : Print_display_array("0110");
                                        break;
                                case '7' : Print_display_array("0111");
                                        break;
                                case '8' : Print_display_array("1000");
                                        break;
                                case '9' : Print_display_array("1001");
                                        break;
                                case 'A' : Print_display_array("1010");
                                        break;
                                case 'B' : Print_display_array("1011");
                                        break;
                                case 'C' : Print_display_array("1100");
                                        break;
                                case 'D' : Print_display_array("1101");
                                        break;
                        }
                        ++count;
                }
}

void Keypad_button_clicked( int row, int column) {
        Print_display(keypad_symbols[row*4+column]);
        data[n_chars] = keypad_symbols[row*4+column];
        ++n_chars;
}

/* USER CODE END 0 */

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

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

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
  // initialization of column pins
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 1);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 1);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 1);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 1);

  Setup_Display();
  Display_example();
  Display_clear();

  /* USER CODE END 2 */

  while (1)
  {
  }
}

// system autogenerated config for project
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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  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();
  }
}

/* USER CODE BEGIN 4 */
// keypad interruption callback function
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

  currentMillis = HAL_GetTick();
  if (currentMillis - previousMillis > 500) {

    /*Configure GPIO pins : PD0 PD1 PD2 PD3 to GPIO_INPUT*/
    GPIO_InitStructPrivate.Pin = GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0;
    GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
    GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
    GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStructPrivate);


    // checking for last 4th column – PD11=1, others - 0
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 1);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);

    if(GPIO_Pin == GPIO_PIN_3 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_3))
    {
       //ASCII value of D
       Display_Write_Ins(GO_TO_START_SECOND_LINE);
       Convert_hex_to_bin();
    }
    else if(GPIO_Pin == GPIO_PIN_2 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_2))
    {
       //ASCII value of C
       Keypad_button_clicked(2, 3);
    }
    else if(GPIO_Pin == GPIO_PIN_1 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_1))
    {
       //ASCII value of B
```

```
        Keypad_button_clicked(1, 3);
    }
    else if(GPIO_Pin == GPIO_PIN_0 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_0))
    {
      //ASCII value of A
        Keypad_button_clicked(0, 3);
    }



    // checking for 3rd column – PD1=1, others - 0
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 1);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);

        if(GPIO_Pin == GPIO_PIN_3 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_3))
        {
                //ASCII value of #
                Display_clear();
                Display_Write_Ins(GO_TO_START_FIRST_LINE);
                n_chars = 0;
        }
        else if(GPIO_Pin == GPIO_PIN_2 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_2))
        {
                //ASCII value of 9
                Keypad_button_clicked(2, 2);
        }
        else if(GPIO_Pin == GPIO_PIN_1 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_1))
        {
                //ASCII value of 6
                Keypad_button_clicked(1, 2);
        }
        else if(GPIO_Pin == GPIO_PIN_0 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_0))
        {
                //ASCII value of 3
                Keypad_button_clicked(0, 2);
        }

        // checking for 2nd column – PD9=1, others - 0
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 1);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);

        if(GPIO_Pin == GPIO_PIN_3 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_3))
        {
                //ASCII value of 0
                Keypad_button_clicked(3, 1);
        }
        else if(GPIO_Pin == GPIO_PIN_2 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_2))
        {
```

```
            //ASCII value of 8
            Keypad_button_clicked(2, 1);
        }
        else if(GPIO_Pin == GPIO_PIN_1 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_1))
        {
            //ASCII value of 5
            Keypad_button_clicked(1, 1);
        }
        else if(GPIO_Pin == GPIO_PIN_0 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_0))
        {
            //ASCII value of 2
            Keypad_button_clicked(0, 1);
        }


        // checking for 1st column – PD8=1, others - 0
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 1);
        if(GPIO_Pin == GPIO_PIN_3 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_3))
        {
            //ASCII value of *
            Display_Write_Ins(GO_TO_START_SECOND_LINE);
            Convert_hex_to_bin();
        }
        else if(GPIO_Pin == GPIO_PIN_2 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_2))
        {
            //ASCII value of 7
            Keypad_button_clicked(2, 0);
        }
        else if(GPIO_Pin == GPIO_PIN_1 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_1))
        {
            //ASCII value of 4
            Keypad_button_clicked(1, 0);
        }
        else if(GPIO_Pin == GPIO_PIN_0 && HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_0))
        {
            //ASCII value of 1
            Keypad_button_clicked(0, 0);
        }

        // resetting pins – putting 1 at all pins
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, 1);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, 1);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 1);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 1);

    /*Configure GPIO pins : PD0 PD1 PD2 PD3 back to EXTI*/
    GPIO_InitStructPrivate.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStructPrivate.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStructPrivate);
```

```
    previousMillis = currentMillis;
  }
 }
/* 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 */
```

drivers/clock_config.c

```
#include "clock_config.h"
#include "stm32f4xx_hal_rcc.h"

void setup_clock_for_GPIO(GPIO_TypeDef* GPIOx, FunctionalState state)
{
	if (state) {
		switch ((int)GPIOx) {
				case (int)GPIOA:{
					__HAL_RCC_GPIOA_CLK_ENABLE();
					break;}
				case (int)GPIOB:{
					__HAL_RCC_GPIOB_CLK_ENABLE();
					break;}
```

```
                        case (int)GPIOC:{
                                __HAL_RCC_GPIOC_CLK_ENABLE();
                                break;}
                        case (int)GPIOD:{
                                __HAL_RCC_GPIOD_CLK_ENABLE();
                                break;}
                        case (int)GPIOE: {
                                __HAL_RCC_GPIOE_CLK_ENABLE();
                                break; }
                        default:
                                break;
                }
        }
        else {
                switch ((int)GPIOx) {
                                        case (int)GPIOA:{
                                                __HAL_RCC_GPIOA_CLK_DISABLE();
                                                break;}
                                        case (int)GPIOB:{
                                                __HAL_RCC_GPIOB_CLK_DISABLE();
                                                break;}
                                        case (int)GPIOC:{
                                                __HAL_RCC_GPIOC_CLK_DISABLE();
                                                break;}
                                        case (int)GPIOD:{
                                                __HAL_RCC_GPIOD_CLK_DISABLE();
                                                break;}
                                        case (int)GPIOE:{
                                                __HAL_RCC_GPIOE_CLK_DISABLE();
                                                break;}
                                        default:
                                                break;
                                }
        }
}
```

drivers/clock_config.h

```
#ifndef DRIVERS_CLOCK_CONFIG_H_
#define DRIVERS_CLOCK_CONFIG_H_

#pragma once
#include <stdint.h>
#include <stm32f4xx.h>
#include <stm32f4xx_hal_rcc.h>

#define M_DIVIDER               (4)     // First divider after PLL source MUX
#define N_MULIPLICATOR          (168)   // Multiplicator after M_DIVIDER
#define P_DIVIDER       (2)     //
#define Q_DIVIDER       (8)     //

volatile uint32_t timing_dalay;
```

```
void setup_clock_for_GPIO(GPIO_TypeDef* GPIOx, FunctionalState state);
#endif /* DRIVERS_CLOCK_CONFIG_H_ */
```

drivers/DISPLAY_WH1602B_4Bit.h

```
#ifndef DRIVERS_DISPLAY_WH1602B_4BIT_H_
#define DRIVERS_DISPLAY_WH1602B_4BIT_H_

#include "clock_config.h"
#include <stdio.h>

#define DISPLAY_DELAY (1)
#define DISPLAY_DELAY_40 (40)

#define DISPLAY_PORT            GPIOE
#define DISPLAY_RS                  GPIO_PIN_7
#define DISPLAY_RW                  GPIO_PIN_10
#define DISPLAY_ENA                 GPIO_PIN_11
#define DISPLAY_DB4                 GPIO_PIN_12
#define DISPLAY_DB5                 GPIO_PIN_13
#define DISPLAY_DB6                 GPIO_PIN_14
#define DISPLAY_DB7                 GPIO_PIN_15
#define DISPLAY_BRIGHTNESS  GPIO_PIN_9 //TIM1_CH1 PWM

#define DISPLAY_BIT_7_MASK      (0x80)
#define DISPLAY_BIT_6_MASK      (0x40)
#define DISPLAY_BIT_5_MASK      (0x20)
#define DISPLAY_BIT_4_MASK      (0x10)
#define DISPLAY_BIT_3_MASK      (0x08)
#define DISPLAY_BIT_2_MASK      (0x04)
#define DISPLAY_BIT_1_MASK      (0x02)
#define DISPLAY_BIT_0_MASK      (0x01)

#define DISPLAY_MAX_CHARACKTERS_COUNT     (32)
#define DISPLAY_MAX_FIRST_LINE            (16) // see documentation (if set to 16
display will shift characters to right)
#define DISPLAY_MAX_SECOND_LINE           (16)


#define BIT_7_MASK          (0x80)
#define BIT_6_MASK          (0x40)
#define BIT_5_MASK          (0x20)
#define BIT_4_MASK          (0x10)
#define BIT_3_MASK          (0x08)
#define BIT_2_MASK          (0x04)
#define BIT_1_MASK          (0x02)
#define BIT_0_MASK          (0x01)

// Display instruction (documentation page 28)
#define EMPTY_INSTRUCTION                       (0x00)
```

```
#define GO_TO_START_FIRST_LINE                  (0x80)
#define GO_TO_START_SECOND_LINE                 (0xc0)

#define ONE_LINE_MODE                           (0x20 |
EMPTY_INSTRUCTION)
#define TWO_LINE_MODE                           (0x20 | BIT_3_MASK)

#define DISPLAY_OFF                                     (0x08 |
EMPTY_INSTRUCTION)
#define DISPLAY_ON                                      (0x08 | BIT_2_MASK)
#define CURSOR_OFF                                      (0x08 |
EMPTY_INSTRUCTION)
#define CURSOR_ON                                       (0x08 | BIT_1_MASK)
#define CURSOR_BLINK_OFF                        (0x08 |
EMPTY_INSTRUCTION)
#define CURSOR_BLINK_ON                                 (0x08 | BIT_0_MASK)

#define DISPLAY_CLEAR                           (BIT_0_MASK)

#define DECREMENT_MODE                                  (0x04 |
EMPTY_INSTRUCTION)
#define INCREMENT_MODE                                  (0x04 | BIT_1_MASK)

#define ENTIRE_SHIFT_OFF                        (0x04 |
EMPTY_INSTRUCTION)
#define ENTIRE_SHIFT_ON                                 (0x04 | BIT_0_MASK)

// end  Display instruction


#define EMPTY_REGISTER          (uint8_t(0x00))
#define SET_D7                  (uint8_t(0x00 & BIT_5_MASK))
#define SET_D6                  (uint8_t(0x00 & BIT_4_MASK))
#define SET_D5                  (uint8_t(0x00 & BIT_3_MASK))
#define SET_D4                  (uint8_t(0x00 & BIT_2_MASK))

void Display_example(void);
void Display_init_GPIO(void);
void Display_RW_pusle(void);
void Display_Write_Ins(char);
void Display_Write_Data(char);
void Display_Write_Data_Array(char *data, uint8_t length);
void Display_Init();
void Display_Write_srt(char *str, uint8_t length, uint8_t first_line_offset, uint8_t
second_line_offset);
void Display_clear(void);
void Display_clear_field(uint8_t size);
void Setup_Display(void);
void Print_display(char);
void Print_display_array(char[4]);
void Setup_first_line(void);
```

```
void Setup_second_line(void);
void Display_Delay(int ms);

#endif /* DRIVERS_DISPLAY_WH1602B_4BIT_H_ */
```

drivers/DISPLAY_WH1602B_4Bit.c

```
#include "DISPLAY_WH1602B_4Bit.h"
#include <string.h>

void Display_Init()
{

	Display_Delay(DISPLAY_DELAY_40);
	Display_Write_Ins(0x03);	//as per documentation set 4-bit mode
	Display_Delay(DISPLAY_DELAY_40);

	Display_Write_Ins(0x03);	//as per documentation set 4-bit mode
	Display_Delay(DISPLAY_DELAY_40);

	Display_Write_Ins(0x03);	//as per documentation set 4-bit mode
	Display_Delay(DISPLAY_DELAY_40);


	Display_Write_Ins(0x02);	//as per documentation set 4-bit mode
	Display_Delay(DISPLAY_DELAY_40);

	Display_Write_Ins(TWO_LINE_MODE);
	Display_Delay(DISPLAY_DELAY);

	Display_Write_Ins(DISPLAY_ON | CURSOR_OFF | CURSOR_BLINK_OFF);
	Display_Delay(DISPLAY_DELAY);

	Display_Write_Ins(DISPLAY_CLEAR);
	Display_Delay(DISPLAY_DELAY);

	Display_Write_Ins(INCREMENT_MODE | ENTIRE_SHIFT_OFF);
	Display_Delay(DISPLAY_DELAY);
}

void Display_Delay(int ms)
{
		timing_dalay = HAL_RCC_GetHCLKFreq()/1000 * ms;
		while(timing_dalay--);
}

void Display_Write_Ins(char instruction)
{
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_RS, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_RW, GPIO_PIN_RESET);
```

```
	instruction & DISPLAY_BIT_7_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB7, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB7,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_6_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB6, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB6,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_5_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB5, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB5,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_4_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB4, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB4,
GPIO_PIN_RESET);

	Display_RW_pusle();
	Display_Delay(DISPLAY_DELAY);

	instruction & DISPLAY_BIT_3_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB7, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB7,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_2_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB6, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB6,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_1_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB5, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB5,
GPIO_PIN_RESET);
	instruction & DISPLAY_BIT_0_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB4, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB4,
GPIO_PIN_RESET);

	Display_RW_pusle();
	Display_Delay(DISPLAY_DELAY);

}

void Display_Write_Data(char data)
{
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_RS, GPIO_PIN_SET);
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_RW, GPIO_PIN_RESET);

	data & DISPLAY_BIT_7_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB7, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB7,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_6_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB6, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB6,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_5_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB5, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB5,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_4_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB4, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB4,
GPIO_PIN_RESET);
```

```
	Display_RW_pusle();
	Display_Delay(DISPLAY_DELAY);

	data & DISPLAY_BIT_3_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB7, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB7,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_2_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB6, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB6,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_1_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB5, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB5,
GPIO_PIN_RESET);
	data & DISPLAY_BIT_0_MASK ? HAL_GPIO_WritePin(DISPLAY_PORT,
DISPLAY_DB4, GPIO_PIN_SET) : HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_DB4,
GPIO_PIN_RESET);

	Display_RW_pusle();
	Display_Delay(DISPLAY_DELAY);
}


void Display_Write_Data_Array(char *data, uint8_t length)
{
	while(length--)
	{
		Display_Write_Data(*data++);
	}
	Display_Delay(DISPLAY_DELAY);
}

void Display_init_GPIO(void)
{
	Display_Delay(400);
	setup_clock_for_GPIO(DISPLAY_PORT, ENABLE);
	GPIO_InitTypeDef DISPLAY_GPIO;
	DISPLAY_GPIO.Mode = GPIO_MODE_OUTPUT_PP;
	DISPLAY_GPIO.Speed = GPIO_SPEED_FREQ_HIGH;
	DISPLAY_GPIO.Pull = GPIO_NOPULL;

	DISPLAY_GPIO.Pin = DISPLAY_RS |  DISPLAY_RW | DISPLAY_ENA |
DISPLAY_DB7 | DISPLAY_DB6 | DISPLAY_DB5 | DISPLAY_DB4;
	HAL_GPIO_Init(DISPLAY_PORT, &DISPLAY_GPIO);
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_GPIO.Pin, GPIO_PIN_SET);
}

//pulse
void Display_RW_pusle(void)
{
	HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_ENA, GPIO_PIN_SET);
    HAL_GPIO_WritePin(DISPLAY_PORT, DISPLAY_ENA, GPIO_PIN_RESET);
}
```

```
void Display_Write_srt(char *str, uint8_t length, uint8_t first_line_offset, uint8_t
second_line_offset)
{

	Display_Write_Ins(GO_TO_START_FIRST_LINE + first_line_offset);

	uint8_t i = 0;

	while(str[i] != '\0' && i < DISPLAY_MAX_FIRST_LINE)
	{
		Display_Write_Data(str[i++]);
	}

	if(i <= length)
	{
		Display_Write_Ins(GO_TO_START_SECOND_LINE + second_line_offset);
		while(str[i] != '\0' && i < DISPLAY_MAX_CHARACKTERS_COUNT)
		{
			Display_Write_Data(str[i++]);
		}
	}
}

void Display_clear(void)
{
	Display_Write_Ins(DISPLAY_CLEAR);
}


void Display_clear_field(uint8_t size)
{
	while(size--)
	{
		Display_Write_Data(' ');
	}
}
void Setup_Display(void)
{
	Display_init_GPIO();
	Display_Init();
	Display_Write_Ins(GO_TO_START_FIRST_LINE);
}

void Setup_first_line() {
	Display_Write_Ins(GO_TO_START_FIRST_LINE);
}

void Setup_second_line() {
	Display_Write_Ins(GO_TO_START_SECOND_LINE);
}

void Print_display(char ch) {
```

```
	Display_Write_Data(ch);
}

void Print_display_array(char array[4]) {
	for (int i=0; i<4; ++i) {
		Display_Write_Data(array[i]);
	}
}

void Display_example(void)
{
	Display_init_GPIO();
	Display_Init();

	Display_Write_Ins(GO_TO_START_FIRST_LINE);
	Display_Write_Data('H');
	Display_Write_Data('e');
	Display_Write_Data('l');
	Display_Write_Data('l');
	Display_Write_Data('o');
}
```

main_com.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "usb_device.h"
#include "gpio.h"

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
uint8_t *data = "Hello World from USB CDC\r\n";
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
int main(void)
{
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */
  Setup_GPIO_for_keypad();
  /* USER CODE END 2 */
  while (1)
  {
```

```
        Keypad_loop();
  }
}
```

keypad.h

```
#include <stdint.h>
#include "usbd_cdc_if.h"
//peripheral register addresses
uint32_t volatile *const pGPIODModeReg  =  (uint32_t*)(0x40020C00);
uint32_t volatile *const pInPutDataReg  =  (uint32_t*)(0x40020C00+0x10);
uint32_t volatile *const pOutPutDataReg =  (uint32_t*)(0x40020C00+0x14);
uint32_t volatile *const pClockCtrlReg  =  (uint32_t*)(0x40023800+0x30);
uint32_t volatile *const pPullupDownReg =  (uint32_t*)(0x40020C00 + 0x0C);

void Delay(void);
void Setup_GPIO_for_keypad(void);
void Convert_ (void);
void Keypad_loop(void);
```

keypad.c

```
#include "keypad.h"
#include <string.h>
void Delay(void)
{
        for(uint32_t i=0 ; i < 1200000 ; i++);
}
void Setup_GPIO_for_keypad() {

          //1.Enable the peripheral clock of GPIOD peripheral
                *pClockCtrlReg |= ( 1 << 3);

          // 2.configure PD0,PD1,PD2,PD3 as output (rows)
                *pGPIODModeReg &= ~(0xFF); //clear
                *pGPIODModeReg |= 0x55;   //set

                // 3. configure PD8 , PD9, PD10, PD11 as input (columns)
          *pGPIODModeReg &= ~(0xFF << 16);

                // 4.Enable internal pull-up resistors for PD8 PD9 PD10 PD11
          *pPullupDownReg &= ~(0xFF << 16);
          *pPullupDownReg |=  (0x55 << 16);
}

void Convert_ () {
                // your implementation
}

//char keypad_symbols[16] = {'1', '2', '3', 'A', '4', '5', '6', 'B', '7', '8', '9', 'C', '*', '0', '#', 'D'};
```

```
const char* const keypad_symbols[16] = {"1\r\n", "2\r\n", "3\r\n", "A\r\n", "4\r\n", "5\r\n",
"6\r\n", "B\r\n", "7\r\n", "8\r\n", "9\r\n", "C\r\n", "*\r\n", "0\r\n", "#\r\n", "D\r\n"};

void Keypad_loop() {
                        for (uint8_t i=0; i<4; ++i) {

                                //make all rows HIGH
                                *pOutPutDataReg |= 0x0f;

                                //make Ri LOW(PD0)
                                *pOutPutDataReg &= ~( 1 << i);

                                 //scan the columns

                                //check C1 of Ri is low or high
                                if(!(*pInPutDataReg & ( 1 << 8))) {
                                        //key is pressed
                                        Delay();
                                        CDC_Transmit_FS((unsigned
char*)keypad_symbols[i*4+0], strlen(keypad_symbols[i*4+0]));
                                        Delay();
                                }

                                //check C2 of Ri is low or high
                                if(!(*pInPutDataReg & ( 1 << 9))) {
                                        Delay();
                                        CDC_Transmit_FS((unsigned
char*)keypad_symbols[i*4+1], strlen(keypad_symbols[i*4+1]));
                                        Delay();

                                }

                                //check C3 of Ri is low or high
                                if(!(*pInPutDataReg & ( 1 << 10))) {
                                        Delay();
                                        CDC_Transmit_FS((unsigned
char*)keypad_symbols[i*4+2], strlen(keypad_symbols[i*4+2]));
                                        Delay();
                                }

                                //check C4 of Ri is low or high
                                if(!(*pInPutDataReg & ( 1 << 11))) {

                                        Delay();
                                        CDC_Transmit_FS((unsigned
char*)keypad_symbols[i*4+3], strlen(keypad_symbols[i*4+3]));
                                        Delay();
                                }
                                //delay();
                        }
}
```