МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з курсу:

«ПРОЕКТУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ»

для студентів ступеня вищої освіти «бакалавр»
спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології»

Тернопіль - 2023

Методичні вказівки до виконання лабораторних робіт з дисципліни «Проектування мікропроцесорних систем» для здобувачів вищої освіти на першому (бакалаврському) рівні за спеціальністю 151 – «Автоматизація та комп’ютерно-інтегровані технології» / Укл.: Заставний О.М. – Тернопіль: ЗУНУ, 2023. – 61с.

Укладач: Заставний Олег Михайлович к.т.н., старший викладач кафедри спеціалізованих комп’ютерних систем Західноукраїнського національного університету

Рецензенти: Пітух І.Р. к.т.н., доцент кафедри спеціалізованих комп’ютерних систем Західноукраїнського національного університету

Івасьєв С.В. к.т.н., доцент кафедри кібербезпеки Західноукраїнського національного університету

Відповідальний за випуск: к.т.н., доцент, завідувач кафедри спеціалізованих комп’ютерних систем Андрій СЕГІН

Розглянуто та схвалено на засіданні кафедри спеціалізованих комп’ютерних систем, протокол № 3 від 12 жовтня 2023р.

Розглянуто та схвалено групою забезпечення спеціальності автоматизація, комп’ютерно-інтегровані технології та робототехніка, протокол №2 від 12.10.2023р.

ЗМІСТ

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

Матеріал лабораторного практикуму служить для допомоги у вивченні можливостей 32-розрядних мікроконтролерів і їх використань в побудові інформаційних систем, організації взаємодії декількох пристроїв між собою, використання передачі та відображення інформації в комп'ютерних системах, навчанні самостійної розробки програмного забезпечення з використанням спеціалізованих програм, проведенні тестування та налагодження на реальних пристроях. У ході виконання лабораторних робіт вивчаються принципи роботи мікроконтролерів, їх основною периферії, організації передачі даних з їх використанням, керування іншими пристроями для вимірювання зовнішніх показників, налаштування відображення інформації, отриманої від зовнішніх пристроїв. Навчають, отримують можливість на практиці самостійно налаштувати роботу демонстраційних прикладів і розробити власні відповідно з індивідуальним завданням.

Матеріал лабораторного практикуму складається з трьох основних розділів:

- Загальний опис архітектури ARM і 32-х розрядних мікроконтролерів STM;
- Загальна інформація, яка необхідна для початку роботи з відлагоджувальною платою STM32F4Discovery;
- Шість лабораторних робіт для вивчення основних можливостей, пристроїв і характеристик плати: ШІМ, АЦП, USART, SPI, DMA, таймери та інше.

Для виконання лабораторних робіт необхідно знання основ теорії цифрової схемотехніки, розробки програмного забезпечення та алгоритмізації, а також знання мови програмування С.

2. СЕРЕДОВИЩЕ РОЗРОБКИ STM32CubeIde

Щоб спростити використання багатофункціонального та енергоефективного сімейства мікроконтролерів STM32, у 2019 році компанія ST додала до екосистеми програмного забезпечення STM32Cube безкоштовний багатофункціональний інструмент розробки STM32: STM32CubeIDE.

Щоб працювати так само добре, як інструменти комерційного інтегрованого середовища розробки (IDE), STM32CubeIDE використовує всі переваги технології Atollic®, постачальника вбудованих інструментів розробки, придбаного компанією STMicroelectronics у 2017 році. Використовуючи стандартні умови відкритого ліцензування, це програмне забезпечення IDE додає STM32- спеціальні функції для спрощення та прискорення вбудованих

конструкцій на основі STM32, включаючи потужні інструменти конфігурації мікроконтролера STM32CubeMX та управління проектами.

Інтегрувавши STM32CubeMX із STM32CubeIDE, ST створив більш потужне середовище розробки. повна екосистема STM32Cube також включає записувач коду STM32CubeProgrammer і сімейство моніторів виконання коду STM32CubeMonitor, а також численні автономні пакети мікропрограми MCU.

Що таке STM32CubeIDE?

STM32CubeIDE — це офіційний безкоштовний інструмент розробки програмного забезпечення від ST, заснований на структурі Eclipse®/CDT, ланцюжку інструментів компіляції GCC і інструменті налагодження GDB, і підтримує додавання функціональних плагінів сторонніх розробників. У той же час STM32CubeIDE інтегрує деякі функції STM32CubeMX і STM32CubeProgrammer.

За допомогою STM32CubeIDE розробники можуть робити все, від вибору мікросхеми, конфігурації проекту, генерації коду до редагування коду, компіляції, налагодження та запису.

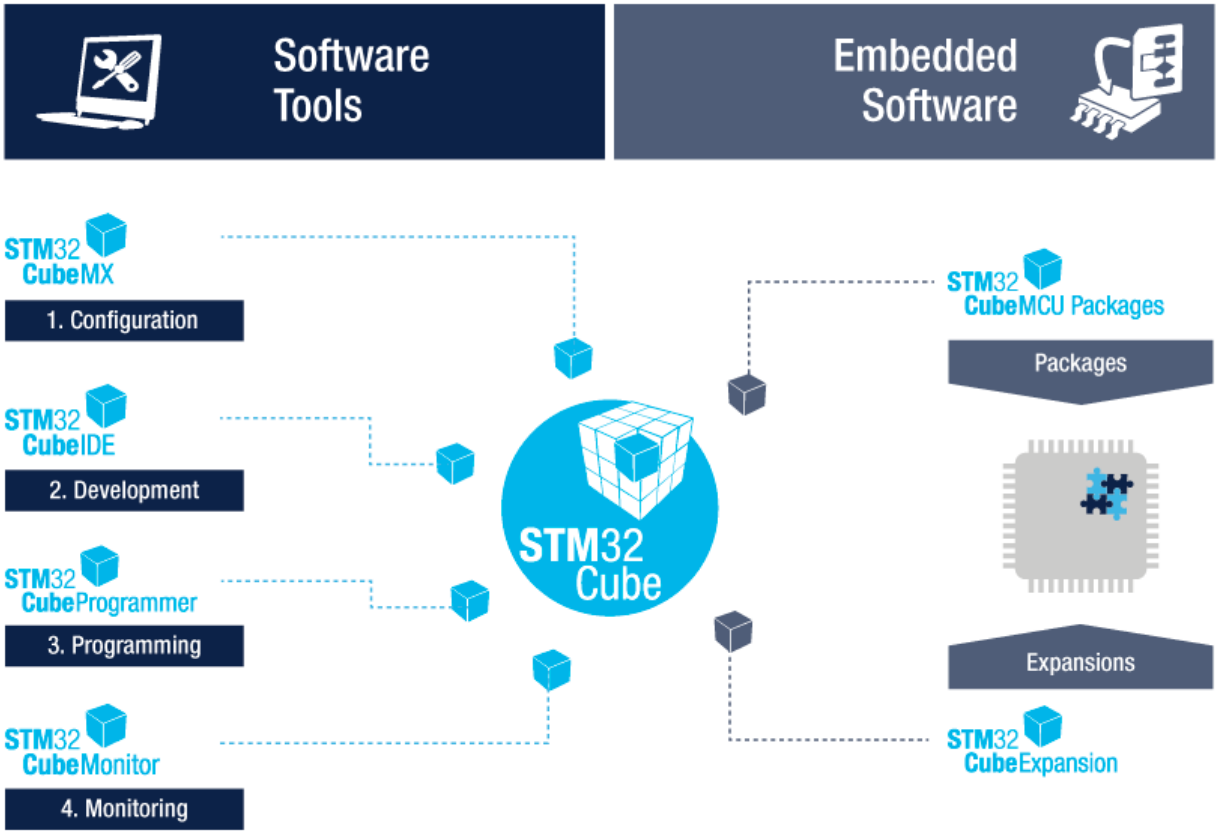

Рис 1.1 - Програмна екосистема STM32Cube

Програмна екосистема STM32Cube

Особливості STM32CubeIDE

1. Інтегруйте STM32CubeMX:

Вибір мікроконтролера STM32

Призначення контактів, годинник, IP та налаштування проміжного ПЗ

Створення проекту та генерація коду ініціалізації

2. На основі Eclipse/CDT, підтримка плагінів Eclipse, використання ланцюжка інструментів ARM і відладчика GDB у GNU C/C++.

3. Інші додаткові функції налагодження:

- Ядра процесора, IP-реєстри та перегляди пам'яті
- Живий змінний вигляд годинника
- Системний аналіз і відстеження в реальному часі (SWV)
- Інструмент аналізу несправностей ЦП

4. Підтримка зонда налагодження ST-LINK і J-Link

5. Підтримка імпорту проектів із TrueSTUDIO та AC6 (SW4STM32)

6. Підтримка операційної системи: Windows, Linux і MacOS

Завантажити STM32CubeIDE

Ось офіційне посилання для завантаження:

https://www.stmicroelectronics.com.cn/en/development-tools/stm32cubeide.html

Робоча область STM32CubeIDE

STM32CubeIDE — це фреймворк на основі Eclipse, який успадковує деякі функції Eclipse, незнайомі користувачам, наприклад перспективні види, робочі області тощо.

Робочий простір : STM32CubeIDE керує проектами через робочий простір. Під час відкриття STM32Cube створюється новий робочий простір за замовчуванням. Новий або імпортований проект належатиме до цієї робочої області. Проекти в одному робочому просторі мають однакову конфігурацію на рівні IDE (встановлену у «Вікно» → «Параметри»), наприклад налаштування стилю відображення та редагування. З точки зору файлової системи, робоча область — це папка, яка містить кілька папок проекту та папку з назвою «.metadata», яка містить інформацію про всі проекти в цій робочій області. метадані» містить інформацію про всі проекти в робочій області. Папка «.metadata» містить інформацію про всі проекти в робочій області. Користувач

може перемикатися між робочими областями за допомогою меню «Файл»→ «Змінити робочу область».

Перспектива : перспектива — це серія вікон, пов'язаних із певним типом функції. Зазвичай використовуються C/C++ Edit Perspective, Debug Perspective і CubeMX Configuration Perspective.

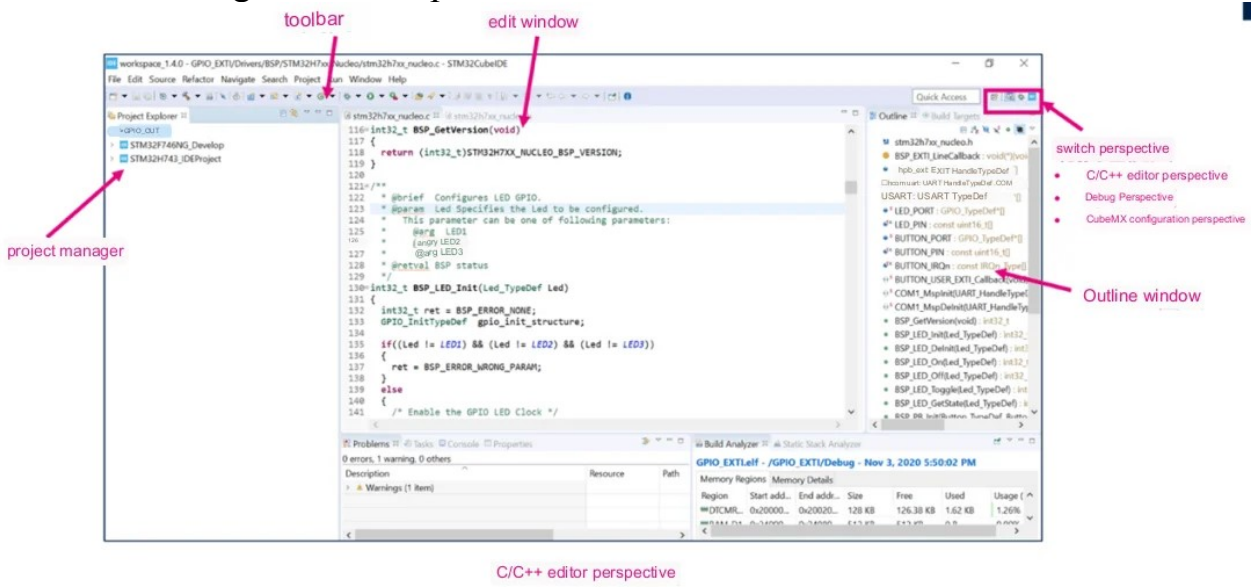

Рис. 1.2 - Перспектива STM32CubeIDE

Перспектива STM32CubeIDE

Перспектива редагування C/C++ включає менеджер проекту, вікно редагування, вікно структури тощо. Менеджер проектів дозволяє переглядати та керувати всіма проектами в поточному робочому просторі. Двічі клацніть, щоб відкрити файл у проекті, і вміст файлу відобразиться у вікні редагування, де його можна редагувати. У крайньому правому вікні «Структура» перелічено всі функції, змінні та макроси, визначені у поточному відкритому файлі.

Меню Window->Show->View дозволяє відкривати та закривати вікна, які потрібно відобразити в перспективі редагування C/C++.

Ви можете перемикатися між різними видами перспективи за допомогою піктограм у верхньому правому куті, наприклад, клацнувши піктограму сканера, ви можете переключитися на перегляд перспективи налагодження. Натискання кнопки «Налагодження» на панелі інструментів під переглядом перспективи редагування C/C++ також автоматично перемикається на перегляд перспективи налагодження після початку налагодження.

Як використовувати STM32CubeIDE?

1. Управління проектами

Створити новий проект

Використовуючи STM32CubeIDE, користувачі можуть почати новий проект різними способами. Екран привітання STM32CubeIDE містить список

точок швидкого входу для створення/імпорту проекту, що відповідає чотирьом сценаріям нижче. Відповідні функції також можна реалізувати за допомогою команди «Створити» та «Імпортувати» в меню «Файл».

1. Створіть новий проект STM32 з нуля
2. Уже існує файл конфігурації STM32CubeMX (файл *.ioc), і ви хочете створити новий проект STM32 на основі цього файлу ioc.
3. У мене вже є проект SW4STM32 або TrueSTUDIO, і я хочу перетворити його на проект STM32CubeIDE.
4. Створіть новий проект на основі процедур у бібліотеці STM32Cube

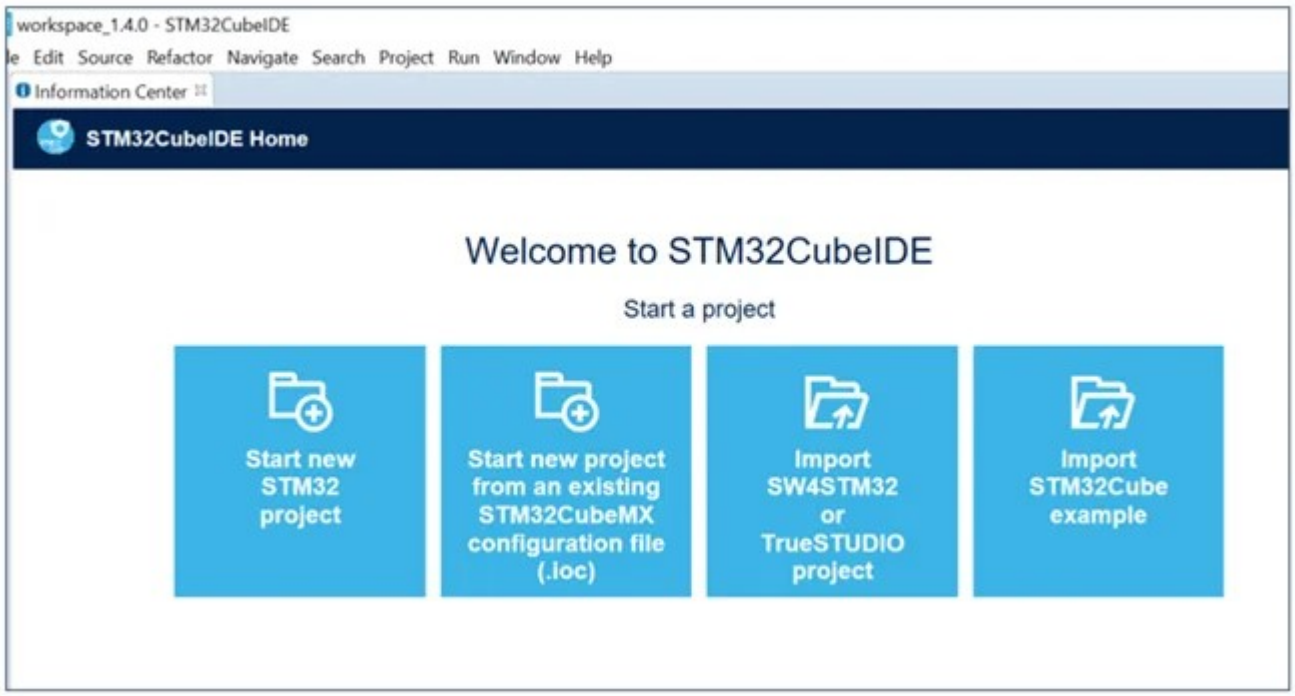

Рис. 1.3 - Новий проект STM32CubeIDE

Новий проект STM32CubeIDE

Структура проекту STM32CubeIDE

STM32CubeIDE має дві структури проекту, як показано на рисунку нижче.

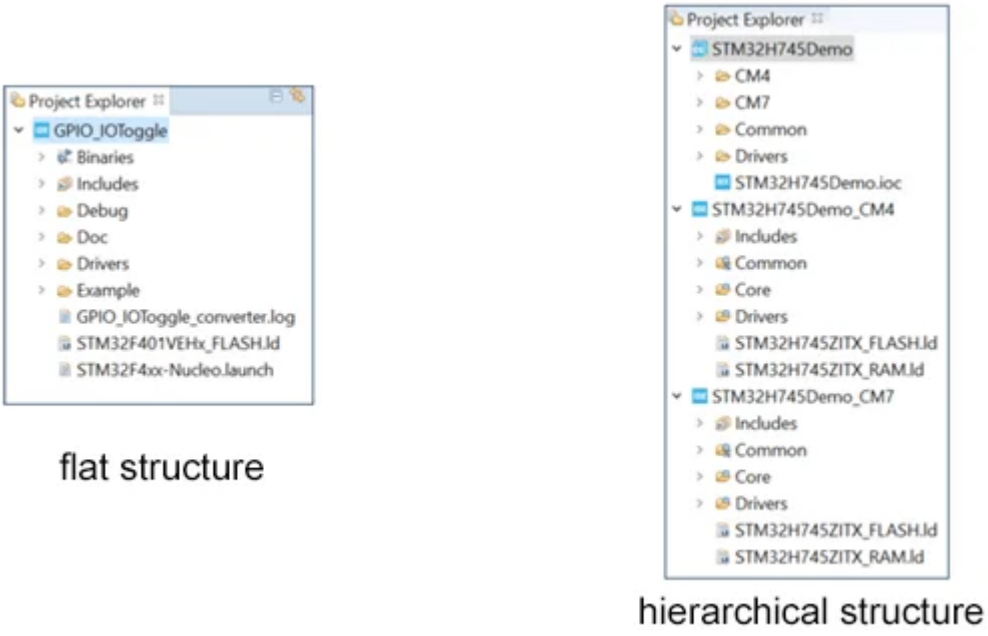

Рис 1.3 - Структурна схема проекту STM32CubeIDE

Одноядерні мікроконтролери є плоскими структурами.

Для двоядерних архітектурних мікроконтролерів або мікроконтролерів безпеки, таких як серії STM32H7, STM32L5 і STM32MP1, проект STM32CubeIDE є ієрархічною структурою. Візьмемо STM32H7 як приклад, після створення або імпорту проекту STM32H7 ви побачите трирівневу структуру проекту на панелі Project Explorer: верхній рівень – це «кореневий» проект, потім є два «дочірні» проекти, що відповідають CM7. і ядра CM4 відповідно. Верхній рівень — це «кореневий» проект, потім є два «підрозділи» проекту, які відповідають ядрам CM7 і CM4, а файли проекту знаходяться під «підрозділами» проектів. Ці два «підрозділові» проекти CM7 і CM4 є реальними проектами, які можна компілювати та налагоджувати, тоді як «кореневий» проект — це лише «контейнер», який містить «підрозділові» проекти CM7 і CM4. AN5361, AN5394, AN5360 і AN5564 описують, як створювати, імпортувати, компілювати та налагоджувати двоядерні проекти STM32H7, STM32L5, STM32MP1 і STM32WL відповідно в STM32CubeIDE.

Керування бібліотекою прошивки

STM32CubeIDE інтегрує деякі функції STM32CubeMX, дозволяючи генерувати новий проект, безпосередньо вибираючи модель чіпа/розробної плати або вибираючи процедуру. драйвер і програмний код, необхідний для створення проекту за допомогою STM32CubeIDE, надходять із бібліотек мікропрограм кожного сімейства STM32.

У Довідка→Керування пакетами вбудованого програмного забезпечення можна керувати всіма бібліотеками прошивки STM32 та іншими плагінами (встановлювати/видаляти бібліотеки прошивки).

Кнопка Install Now дозволяє STM32CubeIDE автоматично завантажувати та встановлювати з мережі, а кнопка From Local дозволяє інсталювати попередньо завантажені бібліотеки прошивки.

Кнопка «Видалити зараз» дозволяє видалити вибрану бібліотеку прошивки.

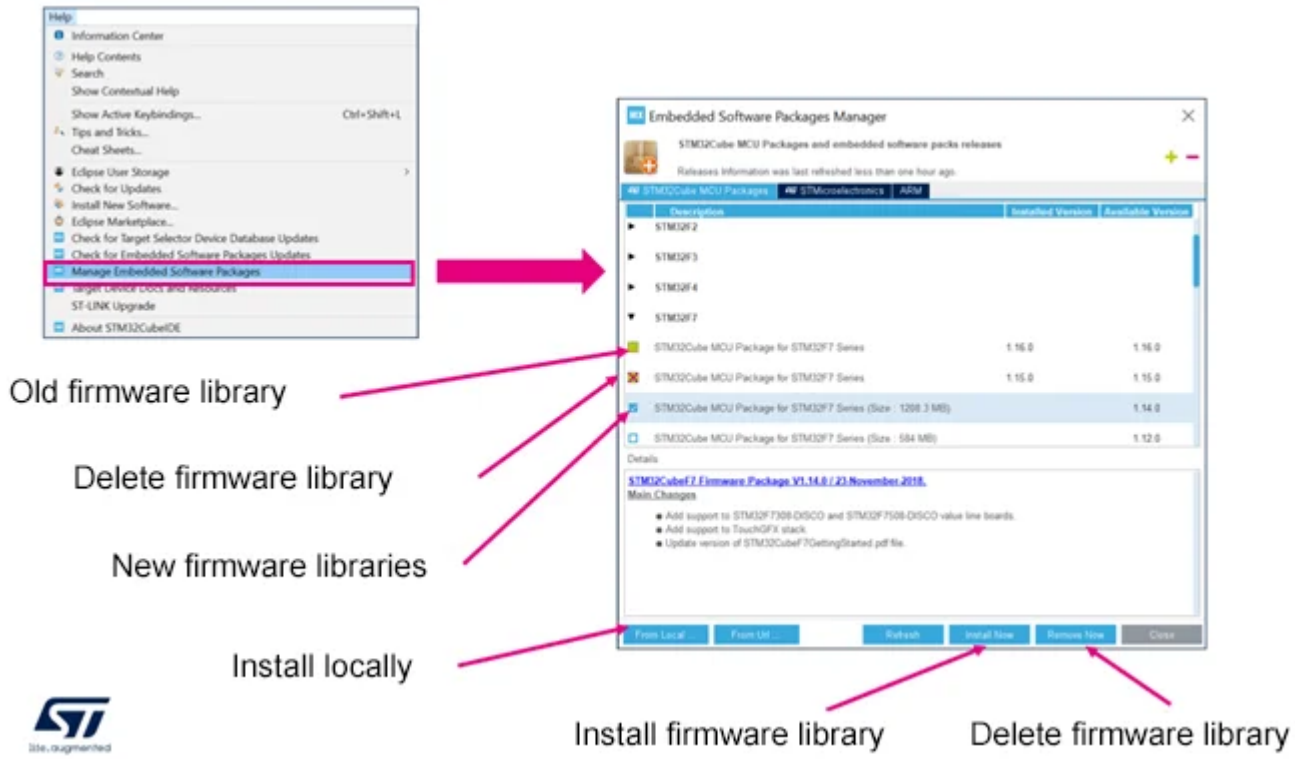

Рис. 1.4 - Бібліотека прошивки STM32CubeIDE

На вкладці STM32Cube Firmware Updater у вікні Window Preferences ви можете встановити шлях до інсталяції мікропрограмної бібліотеки та спосіб її оновлення.

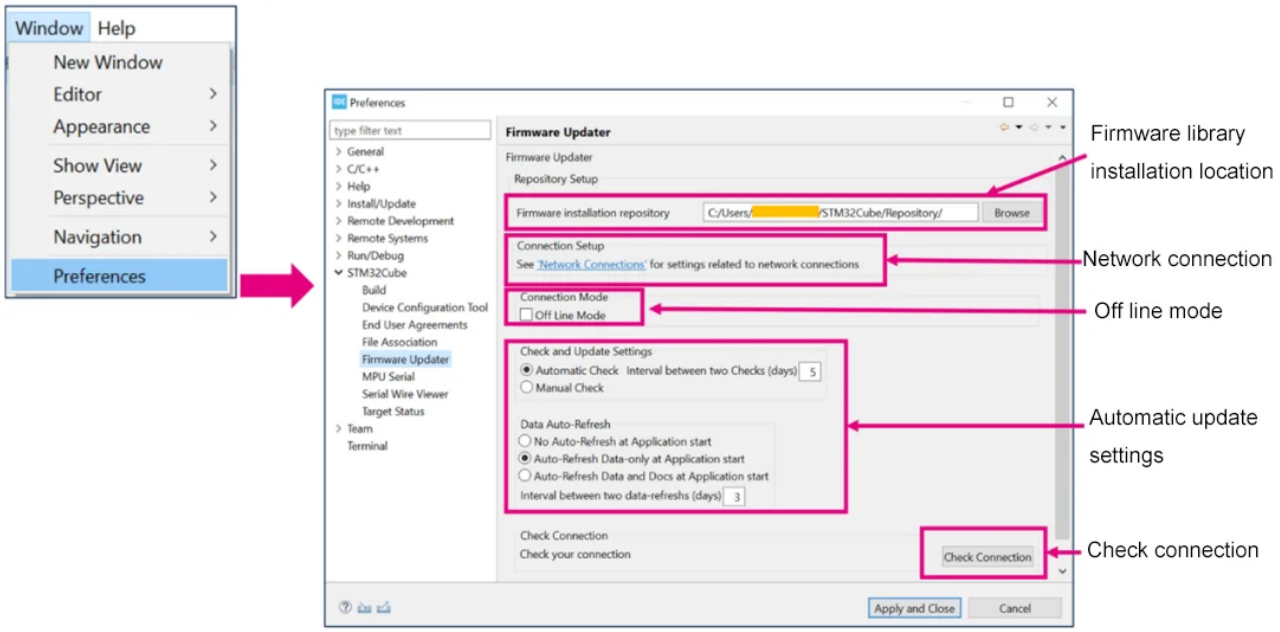

Рис. 1.5 - Налаштування параметрів мікропрограми STM32CubeIDE

Конфігурація мережі

За замовчуванням STM3CubeIDE спробує підключитися до мережі під час відкриття та створення нового проекту. Ви також можете вибрати «Off Line Mode», щоб запобігти підключенню STM32CubeIDE до мережі. Однак вам потрібно інсталювати попередньо завантажені бібліотеки вбудованого програмного забезпечення за допомогою кнопки From Local у вікні Embedded Software Packages Manager на попередньому зображенні, інакше код не буде створено автоматично для нового проекту STM32.

Натисніть кнопку Перевірити підключення, щоб перевірити поточний стан підключення до мережі. Якщо в кінці перевірки з'являється червоний x, це означає, що виникла проблема з конфігурацією мережі, і нам потрібно перейти на сторінку мережевого підключення, щоб її встановити.

На додаток до попереднього активного виявлення стану мережі, якщо завантаження мікропрограми не вдається, також перевірте, чи правильна конфігурація мережі STM32CubeIDE.

Етапи конфігурації показані на наступному рисунку:

1. Перейдіть до меню «Параметри вікна» та виберіть вкладку «Загальні мережеві підключення»

2. Виберіть метод «Вручну»

3. Виберіть «HTTP» і двічі клацніть, щоб відкрити вікно редагування, щоб налаштувати параметри мережевого підключення.

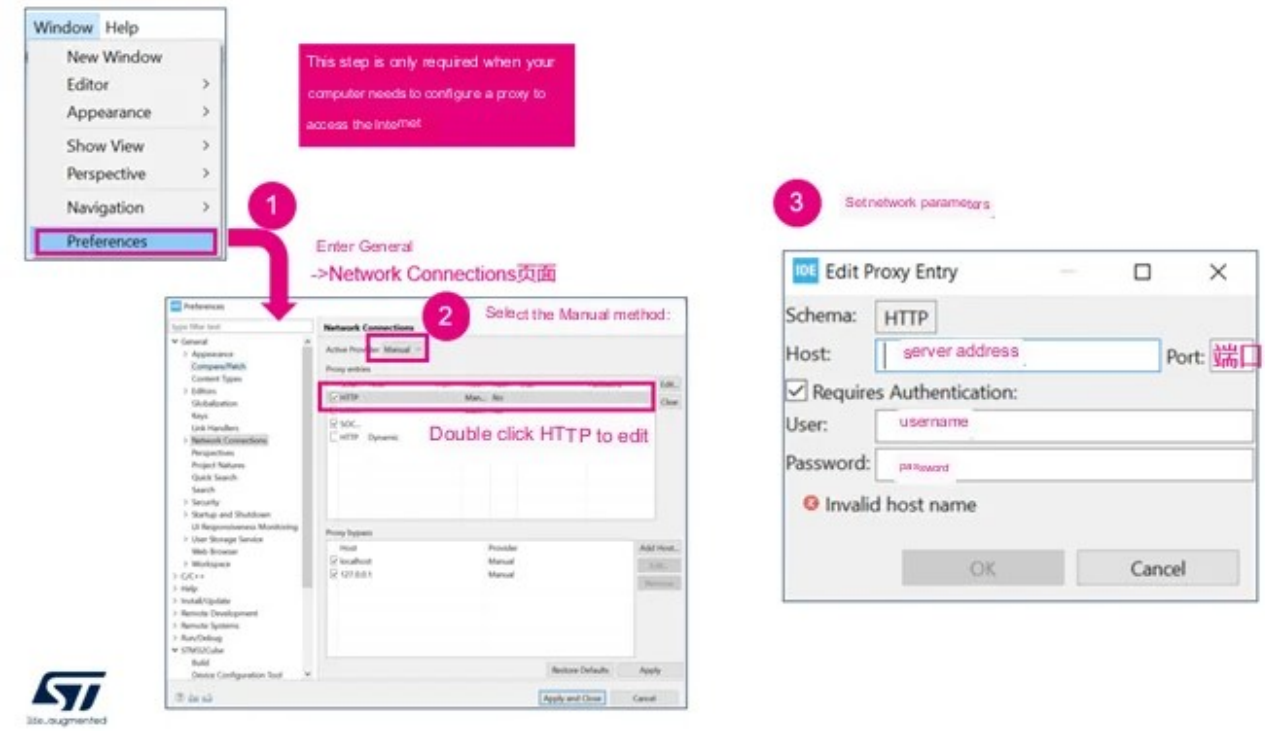

Рис. 1.6 - Конфігурація мережі STM32CubeIDE

Відкрити/закрити/видалити/переключити/експортувати проекти STM32CubeIDE

У вікні Project Explorer ви можете побачити всі проекти в поточній робочій області. Користувачі можуть відкрити/закрити/видалити/імпортувати/експортувати/перейменувати будь-який проект у цьому вікні.

2. Компіляція коду

Налаштування та компіляція властивостей

Виберіть проект у Провіднику проектів, клацніть його правою кнопкою миші та увійдіть у меню властивостей, де ви можете налаштувати елементи компіляції.

Після завершення налаштування проект готовий до компіляції. Ви можете розпочати компіляцію трьома способами:

– Спосіб 1: виберіть проект, клацніть його правою кнопкою миші та виберіть «Створити проект».

– Спосіб 2: виберіть проект, увійдіть у нього з меню «Проект», а потім виберіть «Створити проект».

– Спосіб 3: виберіть проект і клацніть піктограму «Побудувати» на панелі інструментів

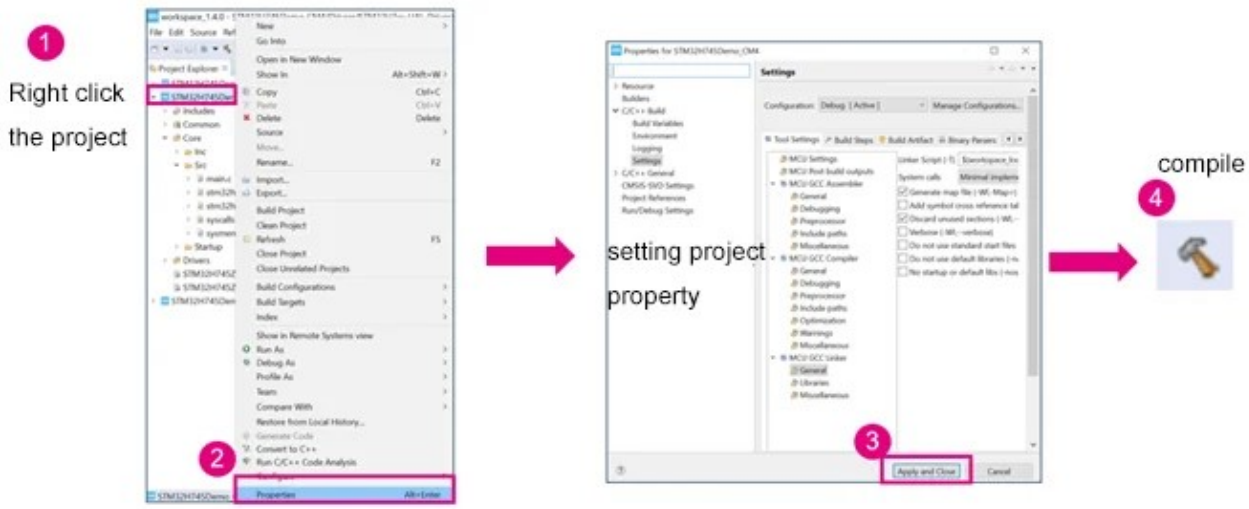

Рис. 1.7 - Конфігурація властивості проекту STM32CubeIDE

Покращення швидкості компіляції

Увімкнувши паралельне збирання, ви можете покращити швидкість компіляції STM32Cube IDE.

Виберіть проект, клацніть його правою кнопкою миші та увійдіть у меню властивостей, виберіть «C/C++ Build» і на вкладці «Поведінка» позначте «Enable parallel build».

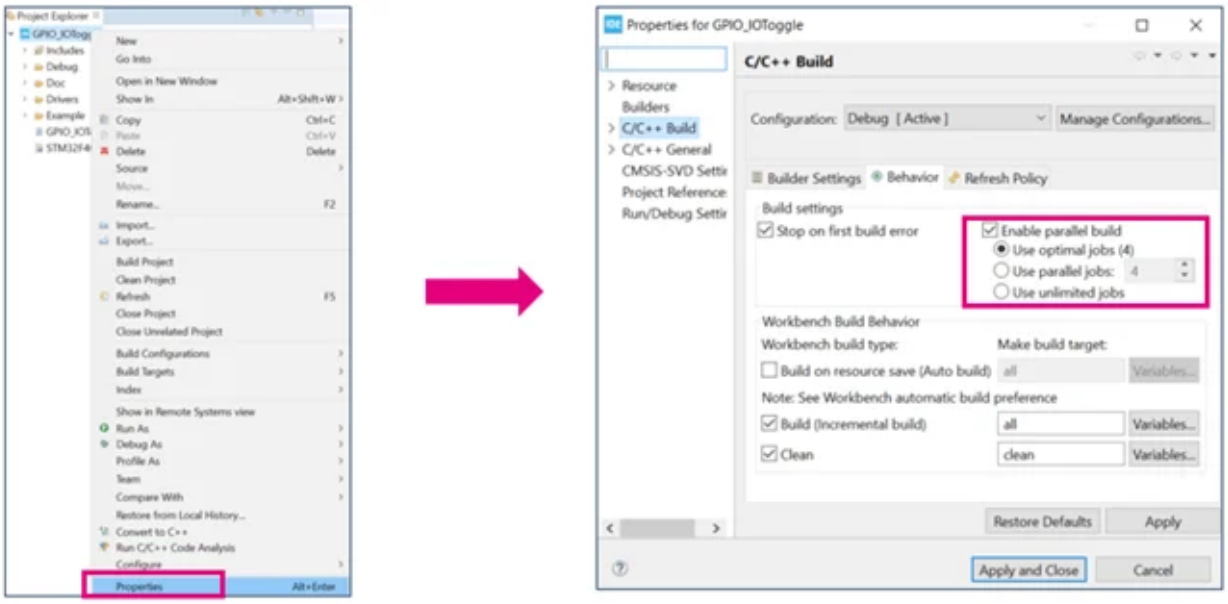

Рис. 1.8 - Паралельна компіляція STM32CubeIDE CC++

Допоміжні інструменти для компіляції

Після компіляції проекту вікно «Аналізатор збірки» показує використання всіх областей пам’яті та розділів, визначених у файлі посилання, включаючи адресу завантаження, адресу запуску, кількість байтів зайнято, скільки байтів залишилося тощо. тощо

Вікно «Static Stack Analyzer» показує використання статичного стеку.

STM32CubeIDE також надає функцію Headless Build, яка дозволяє компілювати з командного рядка, не відкриваючи графічний інтерфейс CubeIDE.

3. Налагодження та запис коду

Налагодити та запустити конфігурацію

Після компіляції проекту STM32CubeIDE без будь-яких помилок він готовий до налагодження та завантаження.

На панелі інструментів C/C++ Perspective є три кнопки, пов’язані із завантаженням і налагодженням: Debug, Run і External Tools.

Маленький трикутник поруч із кнопкою «Налагодження» відкриває меню «Налаштування налагодження», щоб налаштувати параметри налагодження, наприклад вибір налагоджувача, параметри підключення до GDB, параметри ST-LINK, параметри зовнішнього флеш-завантажувача тощо, а також почати налагодження.

За допомогою кнопки «Виконати» ви можете завантажити програму тільки без запуску налагодження.

Кнопка «Зовнішні інструменти» дозволяє викликати зовнішні інструменти командного рядка.

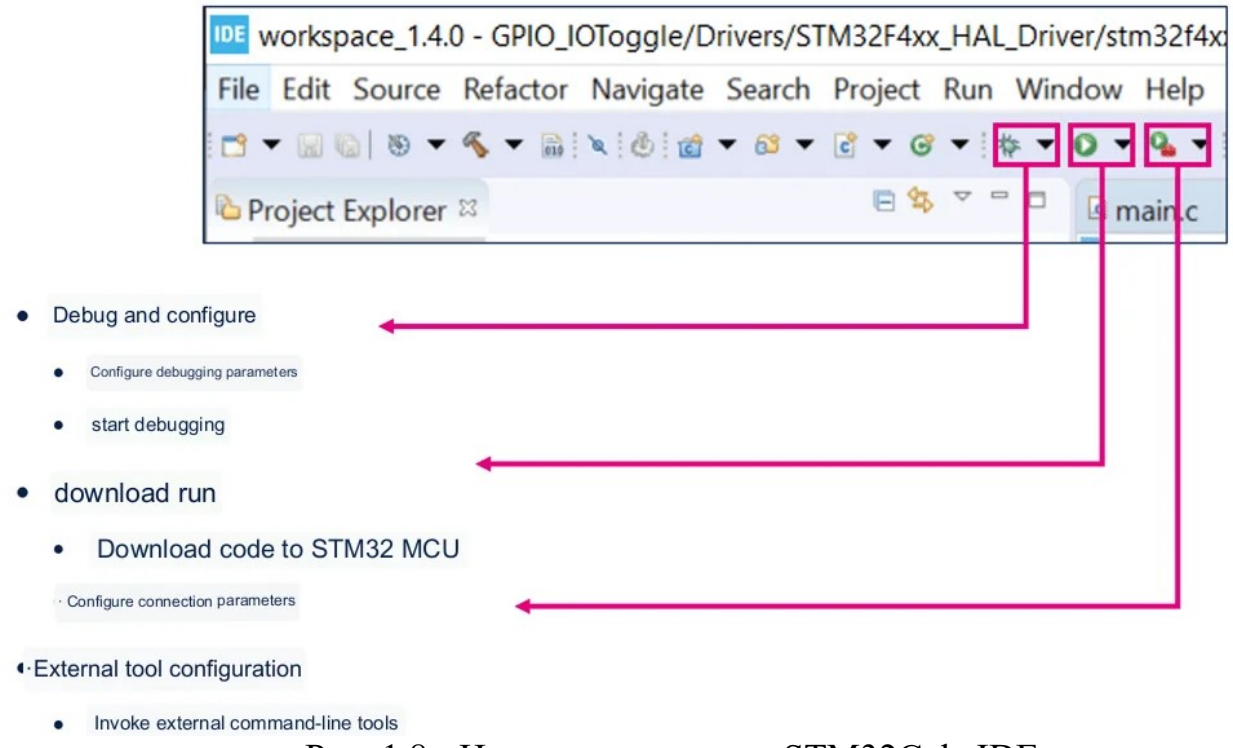

Рис. 1.8 - Налагодження коду STM32CubeIDE

Почніть налагодження

STM32CubeIDE використовує GDB для налагодження, підтримує налагоджувачі STLink і SEGGER J-Link і підтримує підключення до цільового MCU через інтерфейс SWD або JTAG.

Після компіляції проекту STM32CubeIDE налагодження можна розпочати безпосередньо, натиснувши піктограму сканера на панелі інструментів або вибравши меню «Виконати»–>Налагодження.

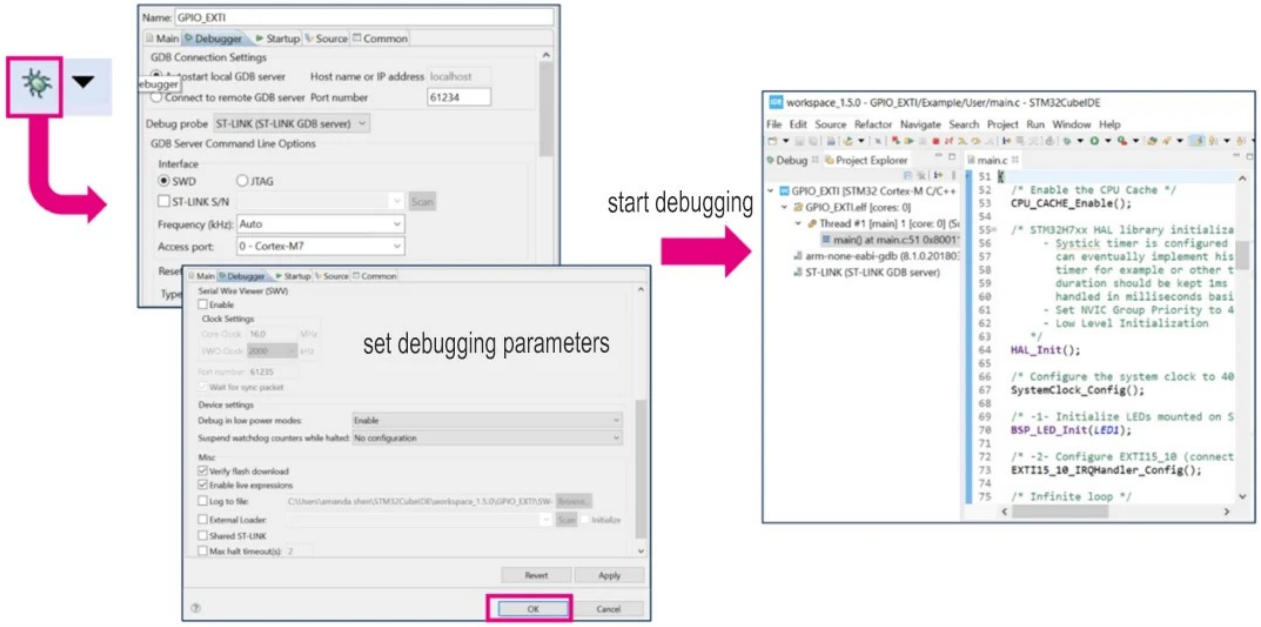

Рис. 1.8 - STM32CubeIDE Розпочати налагодження

Якщо налагодження поточного проекту відбувається вперше, STM32CubeIDE спочатку скомпілює проект, а потім відкриє вікно конфігурації налагодження. Вікно конфігурації налагодження містить параметри вибору інтерфейсу налагодження, налаштувань STLink, скидання налаштувань і налаштувань зовнішнього флеш-завантажувача тощо. Користувачі можуть перевіряти або змінювати конфігурації. Переконавшись, що всі конфігурації правильні, ви можете натиснути OK, щоб розпочати налагодження.

Потім STM32CubeIDE спочатку завантажить програму в MCU, а потім почне виконання із запису програми, зазначеного у файлі посилання (*.ld). Програма за замовчуванням починає виконання з Reset_Handler і зупиняється на першому рядку головної функції, очікуючи наступної інструкції з налагодження.

Основні операції налагодження

Після початку налагодження STM32CubeIDE автоматично переключиться на перспективу налагодження, а кнопки операцій налагодження перераховані на панелі інструментів перспективи налагодження. Як показано на рисунку нижче.

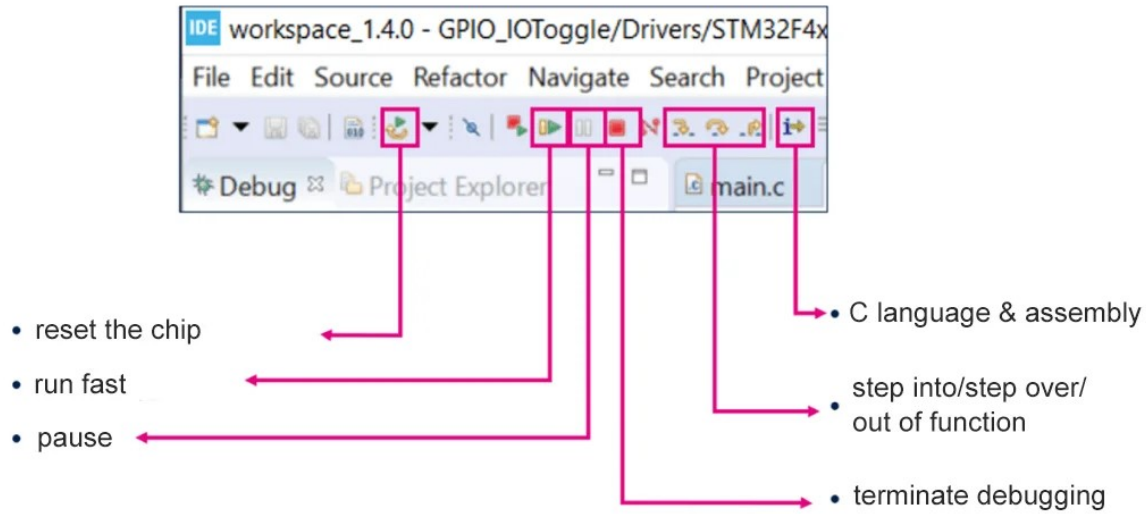

Рис. 1.9 - Основна операція налагодження STM32CubeIDE

4. Використання плагінів

STM32CubeIDE також підтримує плагіни Eclipse. Є 2 способи їх встановлення:

Встановлення за допомогою «Eclipse Marketplace» у меню «Довідка»;

Встановлення за допомогою пункту «Встановити нове програмне забезпечення» в меню «Довідка».

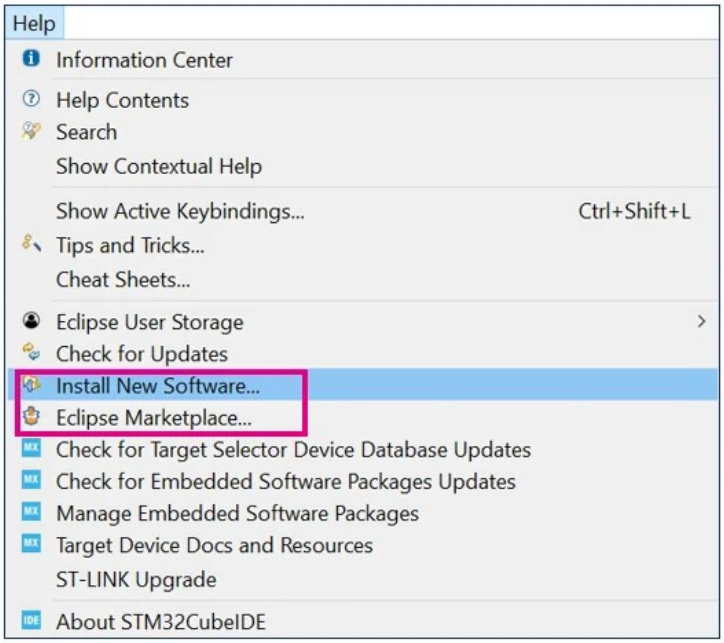

Рис. 1.10 - Встановлення плагінів STM32CubeIDE

Встановлення плагінів STM32CubeIDE

Приклад програмування STM32CubeIDE

Давайте випробуємо процес розробки програмного забезпечення за допомогою STM32CubeIDE за допомогою світлодіодного блимання.

Процедура використовує плату розробки NUCLEO-H743ZI та створює новий проект з нуля через STM32CubeIDE. Тоді проект реалізує просту функцію: вбудований LED1 автоматично перемикається з інтервалом 500 мс.

З цієї процедури можна дізнатися наступне:

- Створіть новий проект STM32CubeIDE
- Налаштуйте чіп STM32
- Додайте код користувача та скомпілюйте
- Встановити параметри налагодження
- Код налагодження (перегляд змінних і регістрів)
- Встановіть контрольні точки

Спочатку виберіть File–>New–>STM32 Project, STM32CubeIDE відкриє вікно вибору MCU. У цьому вікні ви можете вибрати певну модель чіпа, або ви можете вибрати певну плату розробки ST або програму. Тут ми вводимо STM32H743ZI у вікні пошуку, ви можете безпосередньо вибрати цей чіп, а потім натиснути «Далі».

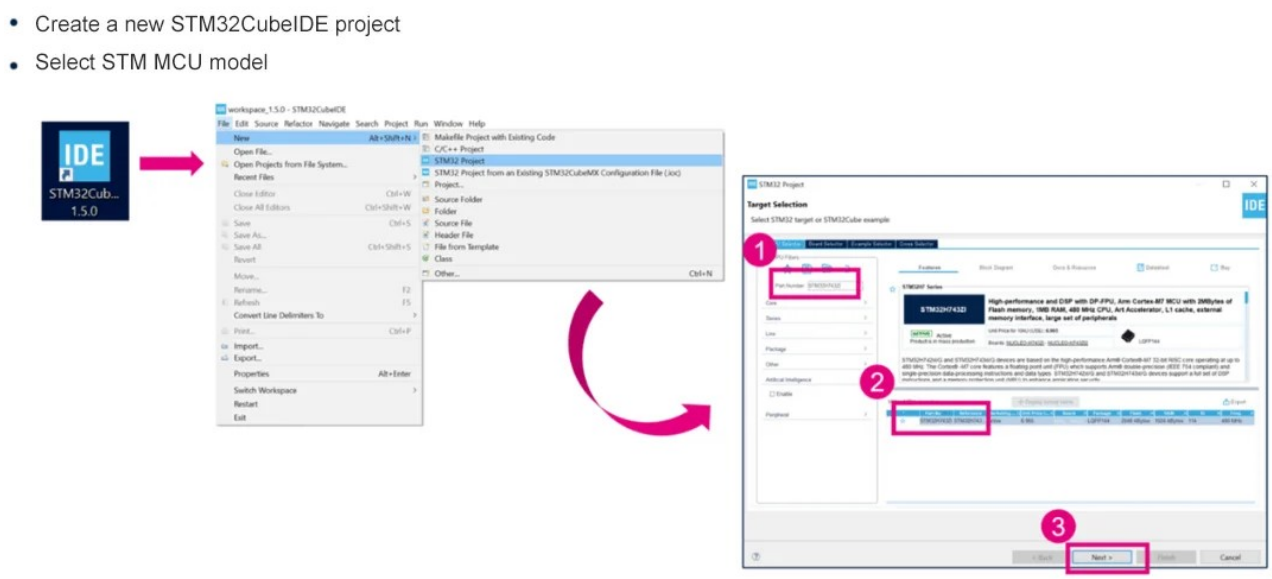

Рис. 1.11 - Створіть новий проект STM32CubeIDE і виберіть модель STM MCU

У наступному спливаючому вікні потрібно вказати назву проекту, тип і місце збереження проекту. За замовчуванням новий проект буде збережено в поточній робочій області, але ви також можете зняти прапорець «Використовувати розташування за замовчуванням» і налаштувати розташування проекту, як показано нижче. Зауважте, що якщо ви спробуєте розмістити два проекти в одному шляху, ви можете отримати помилку під час процесу створення пізніше. Таким чином, ви можете додати назву проекту до шляху, щоб розрізняти різні проекти.

Після завершення налаштування натисніть «Готово», і STM32CubeIDE створить для нас проект і відкриє інтерфейс конфігурації чіпа, який є таким же, як і STM32CubeMX.

Протягом цього часу ви можете отримати спливаюче вікно з повідомленням про те, що ви відкриєте перспективу конфігурації CubeMX, натисніть «Так» і готово.

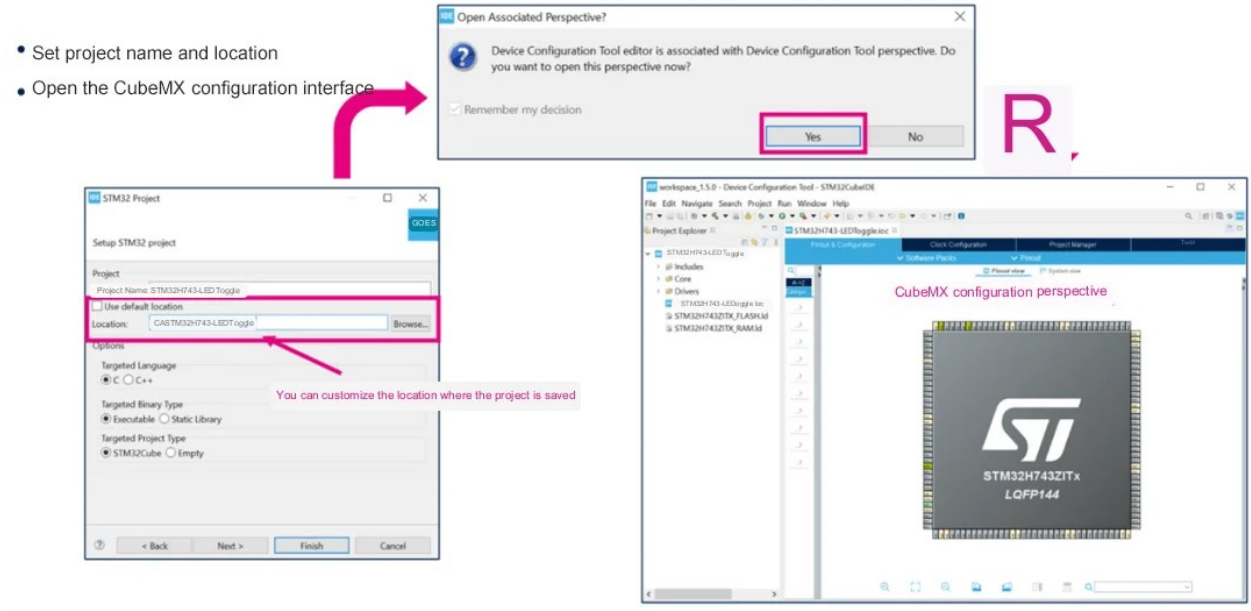

Рис. 1.12 - Перспектива конфігурації STM32CubeMX

Наступне, що потрібно зробити, це налаштувати годинник, GPIO, периферійні пристрої, проміжне програмне забезпечення тощо відповідно до функціональних вимог на екрані конфігурації чіпа. У цій процедурі ми використовуємо лише PB0, який використовується як вихід для керування LED1, тому нам просто потрібно його налаштувати. Годинник просто використовує конфігурацію за замовчуванням.

У режимі перегляду Pinout контакт, який потрібно налаштувати, можна швидко знайти на схемі упаковки мікросхеми за допомогою панелі пошуку.

Виберіть шпильку, клацніть її правою кнопкою миші та виберіть функцію GPIO_Output.

Потім перейдіть до системного перегляду, клацніть модуль GPIO, і тоді ви побачите щойно налаштований контакт PB0. У цьому вікні також можна продовжити налаштування інших параметрів PB0, таких як підтягування/вниз, швидкість тощо. Для нього також можна визначити мітку користувача LDE1, на яку легко посилатися в коді.

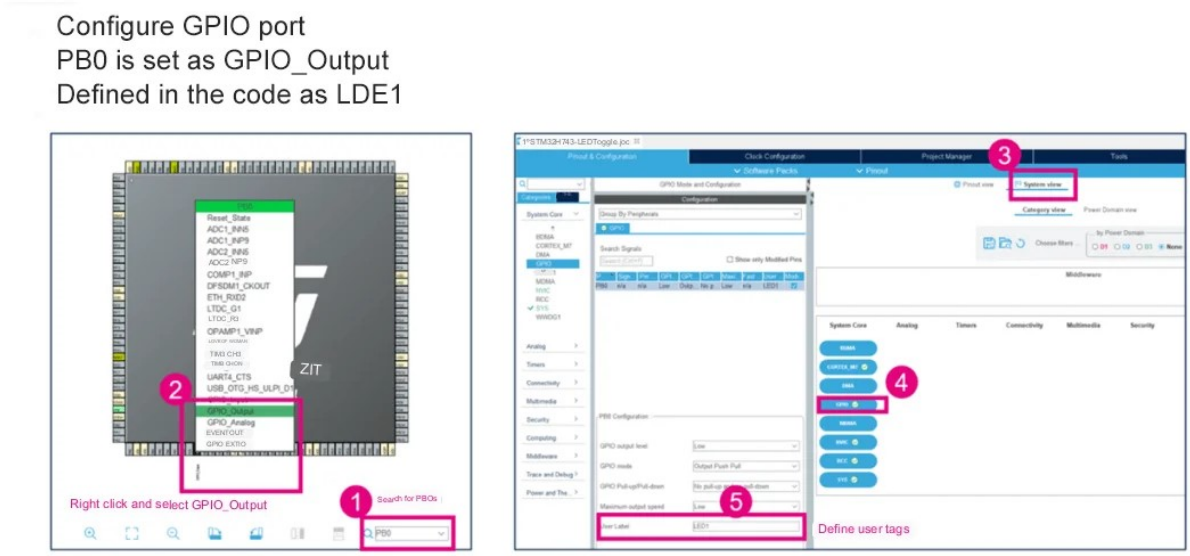

Рис. 1.13 - Налаштуйте GPIO на Create STM32CubeIDE

Після завершення всіх налаштувань виберіть «Проект»–>Створити код, щоб повторно згенерувати код. У цей момент STM32Cube перемикається на перспективний вигляд C/C++, а потім ви можете додати файл користувача або внести зміни до файлу C.

Тут ми додаємо фрагмент коду, який перевертає PB0. Зауважте, що весь код, доданий користувачем, має бути розміщений між оголошеннями «КОРИСТУВАЧА ПОЧАТОК xxx» і «КОД КОРИСТУВАЧА КІНЕЦЬ xxx». Таким чином, коли код буде повторно згенеровано, ця частина коду не буде втрачена.

Після додавання коду натисніть «Побудувати», щоб почати компіляцію.

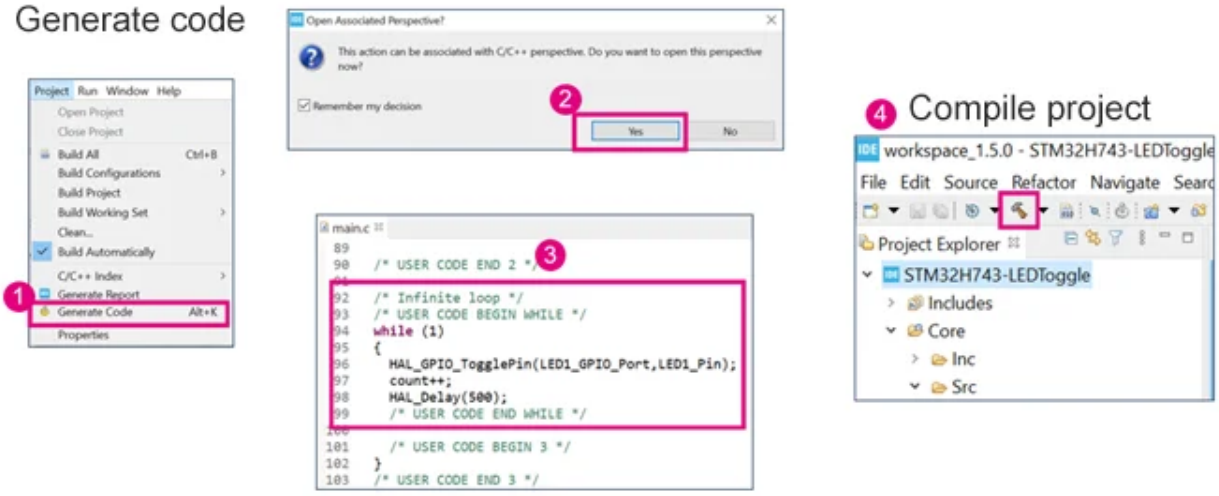

Рис. 1.14 - Створення коду та компіляція проекту

Якщо компіляція завершена і немає повідомлень про помилки. Потім ви можете натиснути Debug, щоб розпочати налагодження.

Під час першого запуску налагодження з'явиться вікно конфігурації параметрів налагодження. Переконайтеся, що всі параметри правильні, натисніть OK, і STM32CubeIDE автоматично переключиться з інтерфейсу редагування на інтерфейс налагодження. Як показано на рисунку нижче.

В інтерфейсі налагодження ви можете налагодити за один крок за допомогою кнопок дій на панелі інструментів.

Двічі клацніть крайню ліву маркерну смугу рядка коду, щоб додати точку зупину в цьому рядку коду.

З правого боку відкриваються кілька вікон налагодження, зокрема: локальні змінні, точки зупину, глобальні змінні та регістри тощо. Ці вікна можна відкрити або закрити в меню «Вікно»–>Показати перегляд.

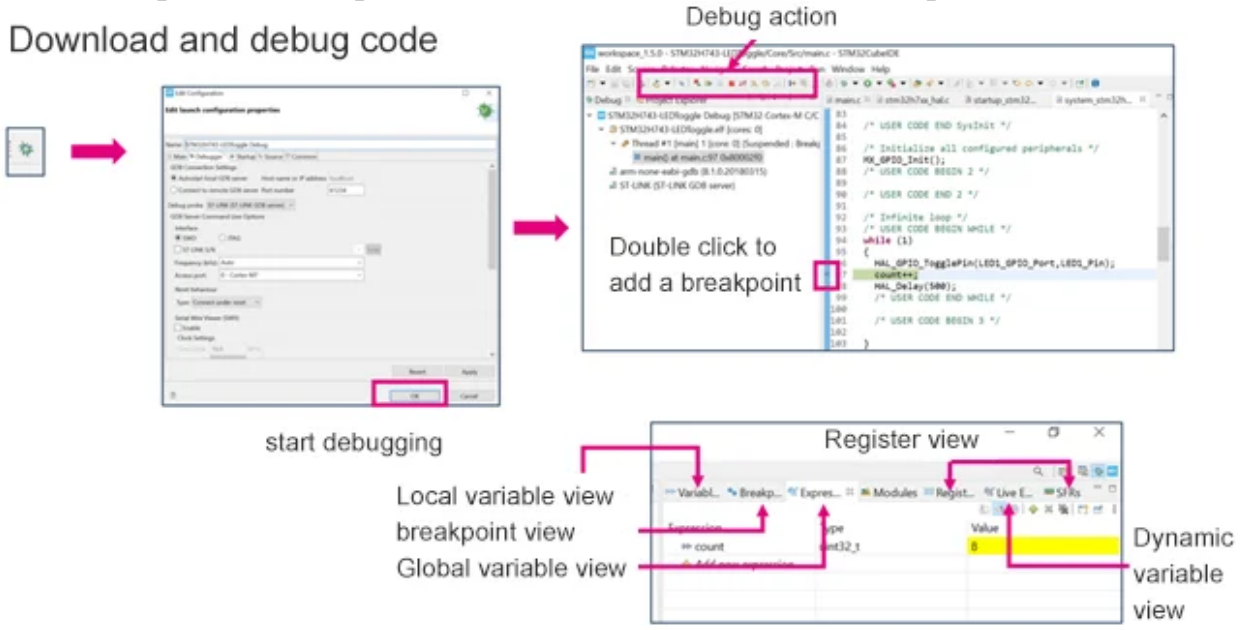

Рис. 1.14 - Код налагодження STM32CubeIDE

Для розробки програмного забезпечення під 32 - розрядні процесори ARM (в тому числі і для розглянутої плати) рекомендуються такі програмні продукти:

- STM32CubeIde;
- IAR Embedded Workbench;
- Keil µVision з інструментами MDK- ARM;

Також є багато інших середовищ,проте багато з них що є комерційними і безкоштовне їх використання можливе лише з обмеженнями за розміром бінарного коду програми або за термінами використання .

STM32CubeIde є безкоштовним для використання, та достатньо розвинутим середовищем розробки, яке на даному етапі нічим не поступається комерційним середовищам розробки, хоча слід відмітити, що воно активно розвивається доволі багато років. Дане середовище вимагає мінімальних налаштувань для початку програмування і налагодження, тому є ідеальним варіантом для початку роботи з мікроконтролерами STM32 зокрема з платою STM32F4 Discovery.

Розробка з використанням STM32CubeIde пов'язана з концепцією репозиторію: всі доступні компоненти і бібліотеки встановлюються за допомогою майстра, виходячи з того, для якого саме процесора створений проект. Для завантаження бібліотек необхідне підключення до мережі Інтернет.

Робота з проектом в середовищі STM32CubeIde розглянута далі в лабораторних роботах.

3. ЛАБОРАТОРНІ РОБОТИ

3. 1 Лабораторна робота № 1. Створення проекту в середовищі розробки. Використання портів вводу/виводу

Мета роботи: ознайомитися зі створенням проектів для плати, розглянути їх структуру; вивчити принципи роботи з портами вводу/виводу і організації їх взаємодії

Обладнання та програмне забезпечення: плата STM32F4 Discovery, середовище розробки STM32CubeIde.

Теоретичний матеріал

Контролер STM32F407VG містить п'ять 16 - розрядних портів вводу/виводу загального призначення, які позначені як GPIOx, де x може мати значення A, B, C, D, E. Кожен порт GPIO має чотири 32-бітових регістри конфігурації (GPIOx_MODER, GPIOx_TYPER, GPIOx_SPEEDR, GPIOx_ORD), два 32-бітових регістри даних (GPIOx_ODR , GPIOx_IDR) і два 32-бітових регістра вибору додаткових функцій (GPIOx_AFRH і GPIOx_AFRL).

Світлодіоди, призначені для програмування, на платі підключені до порту D (рис. 3.1).

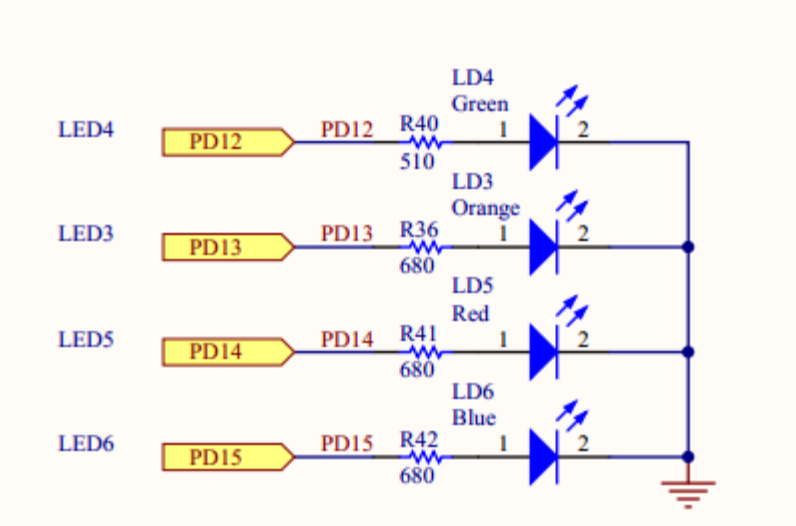

Рис . 3.1 - Схема підключення світлодіодів до порту на платі

Інші чотири світлодіоди виконують службові функції індикації і в програмуванні певних дій не використовуються.

Створення проекту та його структура

Для виконання лабораторних робіт рекомендується встановити середовище розробки STM32CubeIde. Створення проекту виконується за допомогою майстра, який відкривається при виконанні команди меню Project/New Project. У майстрі слід покроково вказати ім'я проекту і його розташування, виробника і МК, для якого призначена програма. Далі робота з проектом здійснюється за допомогою вікна сховища, яке дозволяє додати необхідні бібліотеки управління периферійними частинами МК, а також вікна проекту. Вікно сховища також являє собою майстер, що містить кілька вкладок, основною з яких є вкладка Peripherals.

Проект програми для плати STM32F4 Discovery має схожу структуру для всіх засобів розробки. Зазвичай він включає в себе два внутрішніх каталоги: один - для файлів з бібліотеки CMSIS, а інший - для файлів, призначених для роботи з периферією. Для прикладу представлені типові структури проектів в середовищах розробки CoIDE і Кейл (рис. 3.2).

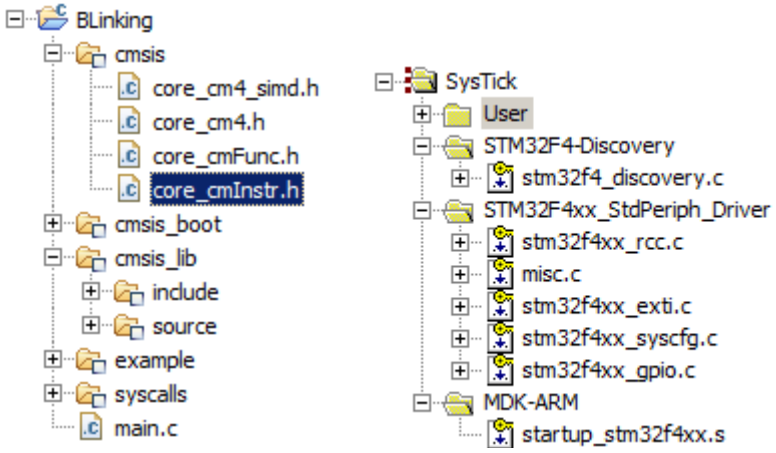

Рис. 3.2 - Структури проектів у різних середовищах розробки

Приклад програми

Розглянемо приклад програми, яка демонструє роботу з портами вводу/виводу. Вона дозволяє перемикати стан світлодіода при натисканні користувальницької кнопки, яка знаходиться на платі.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"

void Delay(uint32_t nCount)
{
	while(nCount--)
	{
	}
}
int main(void)
{
	GPIO_InitTypeDef gpioConf;
	// ініціалізація входу, підключеного до кнопки
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	gpioConf.GPIO_Pin = GPIO_Pin_0;
	gpioConf.GPIO_Mode = GPIO_Mode_IN;
	GPIO_Init(GPIOA, &gpioConf);

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
	// ініціалізація виходу, підключеного до світлодіоду
	gpioConf.GPIO_Pin = GPIO_Pin_13;
	gpioConf.GPIO_Mode = GPIO_Mode_OUT;
	gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
	gpioConf.GPIO_OType = GPIO_OType_PP;
	gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;

	GPIO_Init(GPIOD, &gpioConf);

	while(1) {
		if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
```

```
                    if (GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_13))
                          GPIO_ResetBits(GPIOD, GPIO_Pin_13);
                    else
                          GPIO_SetBits(GPIOD, GPIO_Pin_13);
                    Delay(5000);
                    while(GPIO_ReadInputDataBit(GPIOA,
                    GPIO_Pin_0)==0);
                    Delay(5000);
                  }
        }
}
```

Хід роботи

1 . На основі коду прикладу додатки створити свій проект в середовищі розробки і перевірити його працездатність.

2 . Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.

3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.

4. Реалізувати необхідну функціональність.

5. Запрограмувати плату і продемонструвати роботу програми.

Індивідуальні завдання

У відповідності з варіантом по табл. 3.1 реалізувати схему ввімкнення/вимкнення світлодіодів, розташованих на платі:

Таблиця 3.1. Варіанти ввімкнення світлодіодів

Варіант	1-ий світлодіод	2-ий світлодіод	3-ій світлодіод	4-ий світлодіод
1	1	2	3	4
2	2	1	3	4
3	2	3	1	4
4	2	3	4	1
5	2	4	3	1
6	4	2	1	3
7	3	4	2	1
8	4	3	1	2
9	4	1	3	2
10	1	4	2	3

Номери ввімкнення світлодіодів присвоюються за їх номерами в складі порту D (можна знайти в документації).

3.2 Лабораторна робота № 2.
Переривання та їх використання.
Використання таймерів

Мета роботи: ознайомитись з поняттям перериваннь, можливостями, які вони надають; навчитися використовувати таймери для виконання дій у визначені часові інтервали.

Обладнання та програмне забезпечення: плата STM32F4 Discovery, середовище розробки STM32CubeIde, інструменти побудови проекту GNU інструментарію для ARM Embedded Processors, бібліотека CMSIS при необхідності самостійного підключення файлів до проекту.

Теоретичний матеріал

Переривання - механізм, який дозволяє апаратному забезпеченню повідомляти про настання важливих подій у своїй роботі. У момент, коли відбувається переривання, процесор перемикається з виконання основної програми на виконання відповідного обробника переривань. Як тільки виконання обробника завершено, триває виконання основної програми з місця, в якому вона була перервана.

Для використання переривань необхідно спочатку налаштувати регістр, який називається Nested Vector Interrupt Controller (NVIC), вбудований контролер вектора переривань. Даний регістр є стандартною частиною архітектури ARM і зустрічається на всіх процесорах, незалежно від виробника. NVIC розроблений таким чином, що затримка переривання мінімальна. NVIC підтримує вбудовані переривання з 16-ма рівнями пріоритету.

Мікроконтролер STM32F407VG містить 14 таймерів.

Виробник поділяє всі таймери на три типи:

1) з розширеними можливостями;

2) загального призначення;

3) базові.

Кожен таймер може мати до 4 ліній захоплення/порівняння (саме вони використовуються в режимі генерації ШІМ).

Приклад програми

Дана програма демонструє роботу з перериваннями і з таймерами. У ній реалізовано перемикання світлодіода, який підключений до порту вводу/виводу, через певні інтервали часу.

```
#include «stm32f4xx.h»
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"
```

```
void INTTIM_Config(void);
void GPIO_Config(void);

int main(void)
{
            GPIO_Config();
            INTTIM_Config();

      while(1)
      {
      }
}

void TIM2_IRQHandler(void) {
      if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
            TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
            GPIOD->ODR ^= GPIO_Pin_13;
      }
}

void GPIO_Config(void) {
      GPIO_InitTypeDef gpio_struct;
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
      gpio_struct.GPIO_Pin = GPIO_Pin_13;
      gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
      gpio_struct.GPIO_OType = GPIO_OType_PP;
      gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
      gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
      GPIO_Init(GPIOD, &gpio_struct);
}

void INTTIM_Config(void)
{
      NVIC_InitTypeDef nvic_struct;
      nvic_struct.NVIC_IRQChannel = TIM2_IRQn;
      nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
      nvic_struct.NVIC_IRQChannelSubPriority = 1;
      nvic_struct.NVIC_IRQChannelCmd = ENABLE;

      NVIC_Init(&nvic_struct);

      TIM_TimeBaseInitTypeDef tim_struct;
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
      tim_struct.TIM_Period = 10000 - 1;
      tim_struct.TIM_Prescaler = 168 - 1;
      tim_struct.TIM_ClockDivision = 0;
      tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
      TIM_TimeBaseInit(TIM2, &tim_struct);
      TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
      TIM_Cmd(TIM2, ENABLE);
}
```

Звертає на себе увагу включення додаткового заголовного файлу - misc.h. Саме в ньому описані функції, перерахування, структури для роботи з перериваннями.

Хід роботи

1. На основі коду прикладу додатку створити свій проект в середовищі розробки і перевірити його працездатність.
2. Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.
3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.
4. Реалізувати необхідну функціональність.
5. Запрограмувати плату і продемонструвати роботу програми .

Індивідуальні завдання

1. За допомогою одного з таймерів загального призначення згенерувати затримку тривалістю 1с (на основі даних про робочу частоту). Затримку генерувати на основі переривань таймера. Продемонструвати розрахунки, що підтверджують правильність заданої затримки.
2. Реалізувати за допомогою переривання з переповнення таймера тимчасову затримку з почерговим включенням світлодіодів на платі по колу.

3.3 Лабораторна робота № 3.
Генерація сигналу ШІМ

Мета роботи: ознайомитися з можливостями генерації ШІМ за допомогою демонстраційної плати; навчитися генерувати сигнал за заданими параметрами і використовувати його для управління зовнішніми пристроями.

Обладнання та програмне забезпечення: плата STM32F4 Discovery, середовище розробки STM32CubeIde, інструменти побудови проекту GNU інструментарію для ARM Embedded Processors, бібліотека CMSIS при необхідності самостійного підключення файлів до проекту.

Теоретичний матеріал

Широтно-імпульсна модуляція (ШІМ) - спосіб управління середнім значенням напруги на навантаженні шляхом зміни ширини імпульсів, керованих ключем. В основному, мікроконтролери дозволяють генерувати цифровий ШІМ різної частоти.

Оскільки виведення сигналу ШІМ не є основною функцією пінів, які підключені до світлодіодів, то необхідно виконати відповідну їх конфігурацію. Для цього слід задати виконання пінами альтернативних функцій. Генерація ШІМ у них пов'язана з використанням додаткових режимів таймера.

Перед підключенням пристрою відомі такі параметри ШІМ, як частота і коефіцієнт заповнення. Для їх розрахунку в контроллерах STM32 необхідно визначити значення переддільника і автоматично завантажувати значення в регістрі ARR (Auto-Reload Register). Розрахунок значення, яке слід записати в переддільник виконується таким чином:

$$PSC = \frac{TIMxCLK}{TIMxCNT} - 1,$$

де *PSC* - значення переддільника,
TIMxCLK - вхідна частота роботи таймера,
TIMxCNT - частота лічильника.

Для отримання необхідної вихідної частоти слід записати значення в регістр ARR, яке отримується з наступного співвідношення:

$$ARR\_VAL = \frac{TIMxCNT}{TIMx\_out\_freq} - 1,$$

де *ARR_VAL* - значення для запису в регістр ARR,
TIMxCNT - частота лічильника,
TIMx_ out_freq – необхідна вихідна частота ШІМ.

Останнім етапом є визначення потрібного коефіцієнта заповнення, що забезпечить потрібне значення напруги на виході. Дане налаштування проводиться за допомогою регістра захоплення/порівняння (capture/compare register, CCRx), виходячи з наступного співвідношення:

$$D = \frac{CCRx\_VAL}{ARR\_VAL} * 100\%,$$

де D - коефіцієнт заповнення ,
CCRx_Val - значення в регістрі CCRx (x - номер регістра для конкретної лінії),
ARR_VAL - значення для запису в регістр ARR.

Особливістю даних мікроконтролерів є те, що в переддільник та інші регістри можна записати будь-яке значення, яке можна описати з допомогою відведеної кількості розрядів.

Видача сигналу ШІМ на вихід не є основним режимом роботи виводів порту, а відноситься до додаткових (альтернативних) режимів. Тому попередньо потрібно задати потрібний режим в налаштуваннях порту.

Для підключення альтернативних функцій до портів вводу/виводу необхідно:

1. Підключити пін до альтернативної функції відповідного периферійного пристрою з допомогою функції GPIO_PinAFConfig.
2. Сконфігурувати пін в режим виконання альтернативної функції - GPIO_Mode_AF.
3. Виконати інші налаштування.
4. Викликати GPIO_Init для застосування зазначених налаштувань.

Приклад програми

Для ознайомлення з можливістю генерації сигналу ШІМ і використання його для управління яскравістю світлодіодів представлена наступна програма.

У лістингу представлено зміст файлу з головною функцією.

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
#include "init.h"

int main(void)
{
            init();
```

```
		int brightness = 0;
		int i;
	while(1)
	{
		brightness++;
		TIM4->CCR3 = 333 - (brightness + 0) % 333;
		TIM4->CCR4 = 333 - (brightness + 166 / 2) % 333;
		TIM4->CCR1 = 333 - (brightness + 333 / 2) % 333;
		TIM4->CCR2 = 333 - (brightness + 499 / 2) % 333;

		for(i=0;i < 10000; ++i);
			for(i=0;i < 10000; ++i);
				for(i=0;i < 10000; ++i);
	}
}
```

В основній функції відбувається зміна значень регістру захоплення/порівняння з певною затримкою.

Функції з виконанням налаштувань перенесені в окремий файл init.c. Відповідно, оголошення функцій знаходяться у файлі init.h. У наступному лістингу представлено вміст файлу init.c.

```
#include "init.h"
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
void init() {
	GPIOinit();
	TimerInit();
}
void TimerInit() {
	TIM_TimeBaseInitTypeDef time_init;
	TIM_OCInitTypeDef oc_init;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

	uint16_t PrescalerValue = (uint16_t)((SystemCoreClock / 2) /
21000000);

	time_init.TIM_Period = 665;
	time_init.TIM_Prescaler = PrescalerValue;
	time_init.TIM_ClockDivision = 0;
	time_init.TIM_CounterMode = TIM_CounterMode_Up;

	TIM_TimeBaseInit(TIM4, &time_init);

	oc_init.TIM_OCMode = TIM_OCMode_PWM1;
	oc_init.TIM_OutputState = TIM_OutputState_Enable;
	oc_init.TIM_Pulse = 0;
```

```
        oc_init.TIM_OCPolarity = TIM_OCPolarity_High;

        TIM_OC1Init(TIM4, &oc_init);
        TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

        oc_init.TIM_OutputState = TIM_OutputState_Enable;
        oc_init.TIM_Pulse = 0;

        TIM_OC2Init(TIM4, &oc_init);
        TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);

        oc_init.TIM_OutputState = TIM_OutputState_Enable;
        oc_init.TIM_Pulse = 0;

        TIM_OC3Init(TIM4, &oc_init);
        TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);

        oc_init.TIM_OutputState = TIM_OutputState_Enable;
        oc_init.TIM_Pulse = 0;

        TIM_OC4Init(TIM4, &oc_init);
        TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);

        TIM_ARRPreloadConfig(TIM4, ENABLE);
        TIM_Cmd(TIM4, ENABLE);
    }

        void GPIOinit() {
        GPIO_InitTypeDef gpio_init;
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

        gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 |
GPIO_Pin_15;
        gpio_init.GPIO_Mode = GPIO_Mode_AF;
        gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
        gpio_init.GPIO_OType = GPIO_OType_PP;
        gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOD, &gpio_init);
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4);
        GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
    }
```

Як видно з прикладу, для ініціалізації таймера необхідно відразу дві структури: одна - для ініціалізації безпосередньо таймера, а інша - для задання режиму порівняння виходу. Наведена програма дозволяє послідовно зменшувати яскравість світіння користувальницьких світлодіодів на платі.

Хід роботи

1. На основі коду прикладу програми створити свій проект в середовищі розробки і перевірити його працездатність.

2. Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.

3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.

4. Реалізувати необхідну функціональність.

5. Запрограмувати плату і продемонструвати роботу програми.

3.4 Лабораторна робота № 4.
Використання АЦП

Мета роботи: дослідити можливості використання АЦП

Обладнання та програмне забезпечення: перемички для підключення джерела напруги (батарейки) до входів АЦП

Теоретичний матеріал

Мікроконтролер STM32F407VG включає в себе три АЦП. Розрядність всіх АЦП становить 12 біт. Кожен перетворювач здатний приймати сигнал з шістнадцяти зовнішніх каналів.

Крім того, до складу контролера (не плати) входить сенсор температури. Діапазон вхідних напруг становить 0 - 1,8...3,6В. Сенсор температури підключений до вхідного каналу ADC_IN16, який використовується для того, щоб перетворити вихідну напругу сенсора в цифрове значення.

Внутрішній сенсор температури призначений для відстежування зміни температури, а не для її вимірювання, оскільки зсув показників сенсора може змінюватися в ході змін параметрів процесу. Тому, якщо необхідно точне вимірювання абсолютних значень температури, то для цього краще використовувати зовнішній сенсор.

Приклад програми

У даному прикладі наведена програма, яка дозволяє вимірювати напругу, яка подається на вхід АЦП. Перегляд значення напруги проводиться в режимі налагодження (рис. 3.4).

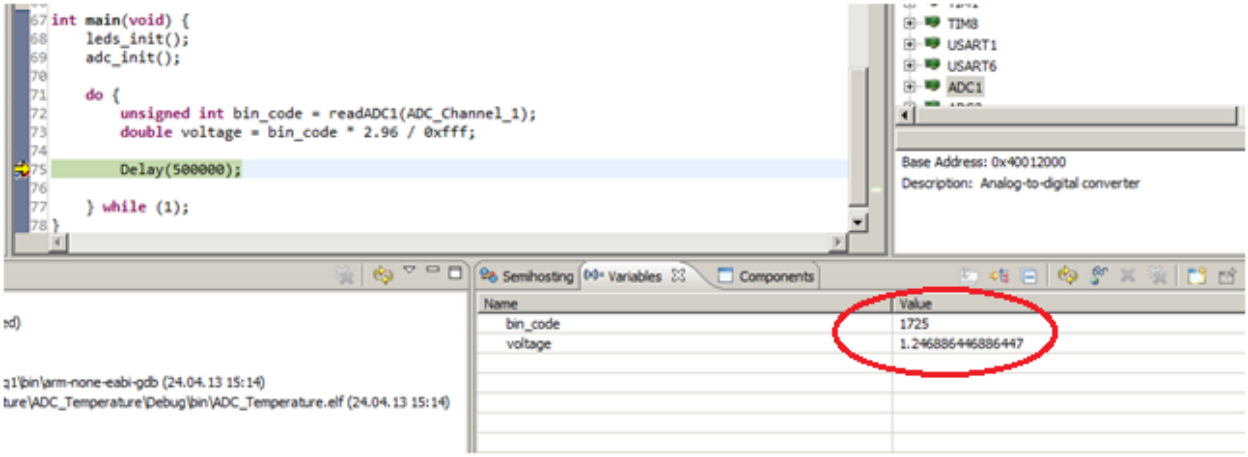

Рис. 3.3 - Перегляд значення напруги в режимі налагодження

Для того, щоб правильно визначити подану напругу, необхідно провести додаткові вимірювання. Для цього за допомогою мультиметра або вольтметра визначаємо напругу, яка відповідає 3В, підключивши їх до відповідних виводів

на платі. Мультиметр показав , що в даному випадку напруга дорівнює 2,96В. Тому записуємо даний коефіцієнт безпосередньо в код програми .

Розрахунок напруги здійснюється за формулою:

$$U_{res} = \frac{U_{ref} * BIN_{res}}{BIN_{max}},$$

де U_{res} - значення поданої напруги,

U_{ref} - напруга, відносно якої проводиться порівняння,

BIN_{max} - двійковий код максимально можливого значення, яке може зберігається в регістрі даних АЦП, залежить від його розрядності (у нашому випадку розрядність АЦП 12, тому максимальній напрузі відповідає число 0xFFF - число, в якого в молодших 12 розрядах всі одиниці),

BIN_{res} - значення двійкового коду з регістра даних АЦП, яке записане після проведення вимірювань.

Відповідно, виходячи з формули 1, при подачі напруги в 0В, ми отримаємо мінімальне значення напруги. Зробимо це з'єднавши виводи землі GND і першого виводу порту А перемичкою. У результаті цього значення в регістрі даних АЦП зміниться наступним чином (рис. 3.4).

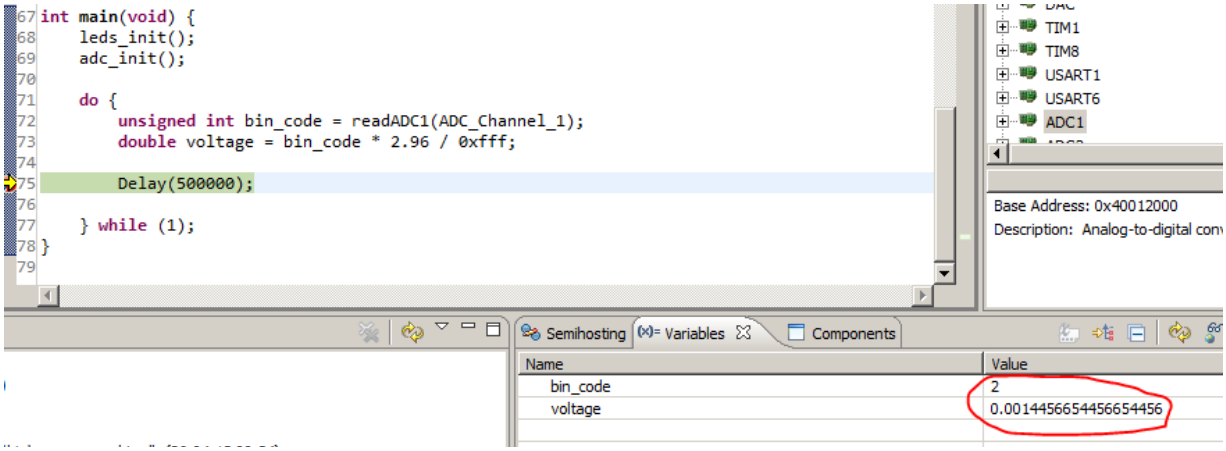

Рис. 3.4 - Перегляд значення напруги в режимі налагодження при підключенні землі до входу

Як бачимо значення досить близьке до нуля, що говорить про правильність роботи програми.

Розглянемо більш детально програму, яка виконує необхідні дії. Вміст основного файлу main.c представлено нижче.

```
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
```

```
#include <stm32f4xx_adc.h>

void leds_init() {
     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
     GPIO_StructInit(&gpio);
     gpio.GPIO_Mode = GPIO_Mode_AN;
     gpio.GPIO_Pin = GPIO_Pin_1;
     GPIO_Init(GPIOA, &gpio);
}

void adc_init() {
     ADC_InitTypeDef ADC_InitStructure;
     ADC_CommonInitTypeDef adc_init;
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
     ADC_DeInit();

     ADC_StructInit(&ADC_InitStructure);
     adc_init.ADC_Mode = ADC_Mode_Independent;
     adc_init.ADC_Prescaler = ADC_Prescaler_Div2;

     ADC_InitStructure.ADC_ScanConvMode = DISABLE;
     ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
     ADC_InitStructure.ADC_ExternalTrigConv                                    =
ADC_ExternalTrigConvEdge_None;
     ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
     ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;

     ADC_CommonInit(&adc_init);
     ADC_Init(ADC1, &ADC_InitStructure);
     ADC_Cmd(ADC1, ENABLE);
}

u16 readADC1(u8 channel) {
     ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_3Cycles);
     ADC_SoftwareStartConv(ADC1);
     while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
     return ADC_GetConversionValue(ADC1);
}

void Delay(unsigned int Val) {
      for (; Val != 0; Val--);
}

int main(void) {
     leds_init();
     adc_init();

     do {
          unsigned int bin_code = readADC1(ADC_Channel_1);
          double voltage = bin_code * 2.96 / 0xfff;
          Delay(500000);
      } while (1);
```

}

У ній є кілька моментів, на які потрібно звернути особливу увагу. По-перше, при ініціалізації порту ми задаємо значення, яке потрібно для його роботи в аналоговому режимі за допомогою значення GPIO_Mode_AN. По-друге, ініціалізація і зчитування значень з АЦП. Ми налаштовуємо, АЦП таким чином, що перетворення виробляється програмно і виконуємо його за допомогою виклику функції readADC1.

Хід роботи

1. На основі коду прикладу програми створити свій проект в середовищі розробки і перевірити його працездатність .

2. Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.

3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.

4. Реалізувати необхідну функціональність.

5. Запрограмувати плату і продемонструвати роботу програми.

Індивідуальні завдання

1. Розробити програму, яка при подачі напруги більш, ніж половина максимальної, вимикає світлодіод, а при подачі більшої напруги включає світлодіод. Підтвердити коректність роботи програми за допомогою мультиметра.

2. Продемонструвати прийом даних з декількох каналів АЦП.

3. Підключити до АЦП для вимірювання вбудований сенсор зміни температури. При зміні в більшу сторону включати червоний світлодіод , при зменшенні - синій.

4. Підключити до АЦП вбудоване джерело напруги. При зміні в більшу сторону включати червоний світлодіод, при зменшенні - синій.

3.5 Лабораторна робота № 5.
Використання USART

Мета роботи: навчитися використовувати універсальний синхронний/асинхронний приймач, організовувати передачу даних інших пристроїв.

Обладнання та програмне забезпечення: плата STM32F4 Discovery, середовище розробки STM32CubeIde.

Теоретичний матеріал

Мікроконтролер на демонстраційній платі містить в загальній складності 6 передавачів. При цьому 4 з них є синхронно-асинхронними (USART1, USART2, USART3, USART6) і 2 тільки асинхронними (UART4, UART5).

Розглянемо процес налаштування USART для прийому даних. Для цього необхідно виконати наступні дії:

1. Включити тактування USART і порти виходів, які використовуються для роботи USART.
2. Виконати налаштування відповідних виходів, вказавши при цьому режим роботи виконання альтернативної функції.
3. Підключити виконання альтернативної функції до зазначених виходів.
4. Задати налаштування роботи USART (частота, розмір передачі, кількість стоп-біт , перевірка на парність).
5. Включити переривання прийому даних для USART.
6. Включити USART.

Приклад програми

У даному прикладі розглянемо прийом даних за допомогою USART1. Для передачі і прийому використовуються виходи порту A під номерами 9 (передача) і 10 (прийом). Для кожного передавача виділені свої пари виводів для прийому і передачі, які можна знайти в документації до мікроконтролера. Насамперед, включаємо тактування потрібних пристроїв.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

Далі проводимо ініціалізацію виводів порту і призначаємо виконання альтернативної функції:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Далі вибираємо альтернативну функцію, виконувану виводами порту. Робимо це за допомогою функції GPIO_PinAFConfig ().

```
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
```

Слід звернути увагу, що налаштуванню підлягають тільки певні виводи, інші, наприклад, PA2 і PA3, використовувати в якості виводів USART не вдасться.

Далі необхідно ініціалізувати USART. Код ініціалізації дуже схожий на ініціалізацію порту.

```
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx;
USART_Init(USART1, &USART_InitStruct);
```

На останньому етапі ініціалізації необхідно налаштувати переривання і включити USART.

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
USART_Cmd(USART1, ENABLE);
```

Після того як ми включили переривання необхідно визначити обробник цього переривання. Обробник переривання має ім'я USART1_IRQHandler і він не повертає і не приймає ніяких даних. Крім того, USART підтримує кілька переривань і для всіх переривань використовується один обробник, тому необхідно перевіряти прапорець (RXNE) в регістрі статусу, щоб організувати обробку різних переривань в різних блоках коду обробника. Після перевірки стану прапорця є можливість скинути прапорець RXNE. Далі зчитуємо дані з регістра даних і виводимо їх на дисплей.

```
void USART1_IRQHandler(void){
      if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
            USART_ClearITPendingBit(USART1, USART_IT_RXNE);
            static uint8_t cnt = 0;
            uint8_t t = USART_ReceiveData(USART1);
            write_char(t);
      }
}
```

Важливим моментом є те, що необхідно задати однакові параметри повідомлення як на пристрої - передавачі, так і на приймачі.

Хід роботи

1. На основі коду прикладу програми створити свій проект в середовищі розробки і перевірити його працездатність.
2. Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.
3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.
4. Реалізувати необхідну функціональність.
5. Запрограмувати плату і продемонструвати роботу програми.

Індивідуальні завдання

1. Задіяти два модулі USART на платі і передати дані з одного в інший в асинхронному режимі. Для демонстрації прийому даних змінювати стан одного з користувацьких світлодіодів на протилежний.
2. Продемонструвати прийом/передачу в синхронному режимі. Для демонстрації прийому даних змінювати стан одного з користувацьких світлодіодів на протилежний.

3.6 Лабораторна робота № 6.
Робота з SPI

Мета роботи: навчитися використовувати інтерфейс SPI для організації передачі даних між пристроями, дослідження режиму ведучого і підлеглого.

Обладнання та програмне забезпечення: плата STM32F4 Discovery, середовище розробки STM32CubeIde, інструменти побудови проекту GNU Toolchain для ARM Вбудовані процесори, бібліотека CMSIS, набір перемичок (3 шт.).

Теоретичний матеріал

Послідовний периферійний інтерфейс (SPI) - популярний інтерфейс для послідовного обміну даними між мікросхемами. Шина SPI організована за принципом «ведучий-підлеглий». В якості ведучої шини зазвичай виступає мікроконтролер, але ним також може бути програмована логіка, DSP -контролер або спеціалізована IC. Підключені до ведучої шини зовнішні пристрої утворюють підлеглі шини. В їх ролі виступають різного роду мікросхеми, в т.ч. запам'ятовуючі пристрої (EEPROM, Flash -пам'ять, SRAM), годинник реального часу (RTC), АЦП/ЦАП, цифрові потенціометри, спеціалізовані контролери та ін.

Головним складовим блоком інтерфейсу SPI є звичайний зсувний регістр, сигнали синхронізації вводу/виводу бітового потоку якого й утворюють інтерфейсні сигнали. Таким чином, протокол SPI правильніше назвати НЕ протоколом передачі даних, а протоколом обміну даними між двома зсувними регістрами, кожен з яких одночасно виконує і функцію приймача, і функцію передавача. Неодмінною умовою передачі даних по шині SPI є генерація сигналу синхронізації шини. Цей сигнал має право генерувати тільки ведуча шина і від цього сигналу повністю залежить робота підлеглої шини.

Приклад найбільш простого підключення по шині SPI зображений на рисунку 3.5. Однойменні виводи з'єднуються між собою. Наявні два канали передачі даних (MOSI і MISO) один канал для подачі тактових імпульсів (SCLK), а також лінія включення веденого пристрою (SS). Підлеглий стає активним при подачі низького рівня по лінії SS.

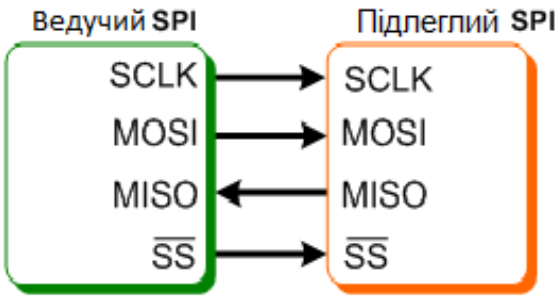

Рис. 3.5 - Схема простого підключення з використанням SPI

SPI - надзвичайно простий і поширений послідовний інтерфейс передачі даних, який базується на регістрах зсуву. Його перевагою в порівнянні з USART є можливість підключення декількох ведених пристроїв. Однак, у порівнянні з USART, він підтримує тільки синхронну передачу. Наявність декількох модулів SPI у складі мікроконтролера дозволяє обійтися без підключення додаткових пристроїв з таким інтерфейсом, а використовувати декілька пристроїв на платі, з'єднавши їх входи між собою перемичками. Цей варіант є оптимальним для навчальних цілей, оскільки вимагає мінімум додаткового обладнання для початку роботи.

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_spi.h>
#include <misc.h>

#define SPI_PINS GPIO_Pin_5 | \
                 GPIO_Pin_6 | \
                 GPIO_Pin_7
void init(void);
void delay(int count);
void SPI1_IRQHandler(void) {
      int res;
      if (SPI_I2S_GetITStatus(SPI1, SPI_I2S_IT_RXNE) != RESET) {
            SPI_I2S_ClearITPendingBit(SPI1, SPI_I2S_IT_RXNE);
            res = SPI_I2S_ReceiveData(SPI1);
      }
}

int main(void)
{
      init();
      int sendData = 0;
   while(1)
   {
     delay(100);
```

```
      SPI_I2S_SendData(SPI2, sendData);
      while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);
      sendData++;
      if (sendData == 0xff) sendData = 0;
    }
}

void init(void) {
      GPIO_InitTypeDef gpio_init;
      SPI_InitTypeDef spi_init;
      NVIC_InitTypeDef nvic_init;

      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA |
            RCC_AHB1Periph_GPIOB |
            RCC_AHB1Periph_GPIOC, ENABLE);

      /* Configure slave pins */
      gpio_init.GPIO_Mode = GPIO_Mode_AF;
      gpio_init.GPIO_Pin = SPI_PINS;
      gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
      gpio_init.GPIO_OType = GPIO_OType_PP;
      gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;

      GPIO_Init(GPIOA, &gpio_init);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
      GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);

      gpio_init.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
      GPIO_Init(GPIOC, &gpio_init);

      GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_SPI2);
      GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_SPI2);

      gpio_init.GPIO_Pin = GPIO_Pin_10;
      GPIO_Init(GPIOB, &gpio_init);

      GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_SPI2);

      RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
      SPI_I2S_DeInit(SPI1);
      spi_init.SPI_Mode = SPI_Mode_Slave;
      spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
      spi_init.SPI_DataSize = SPI_DataSize_8b;
      spi_init.SPI_CPOL = SPI_CPOL_Low;
      spi_init.SPI_CPOL = SPI_CPHA_1Edge;
      spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
      spi_init.SPI_NSS = SPI_NSS_Soft;
      spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
      SPI_Init(SPI1, &spi_init);
      SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);
```

```
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
	SPI_StructInit(&spi_init);
	SPI_I2S_DeInit(SPI2);
	spi_init.SPI_Mode = SPI_Mode_Master;
	spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
	spi_init.SPI_DataSize = SPI_DataSize_8b;
	spi_init.SPI_CPOL = SPI_CPOL_Low;
	spi_init.SPI_CPOL = SPI_CPHA_1Edge;
	spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
	spi_init.SPI_NSS = SPI_NSS_Soft;
	SPI_Init(SPI2, &spi_init);

	nvic_init.NVIC_IRQChannel = SPI1_IRQn;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
	nvic_init.NVIC_IRQChannelSubPriority = 0;
	NVIC_Init(&nvic_init);

	SPI_Cmd(SPI1, ENABLE);
	SPI_Cmd(SPI2, ENABLE);
}

void delay(int count) {
	while(--count);
}
```

Хід роботи

1. На основі коду прикладу програми створити свій проект в середовищі розробки і перевірити його працездатність .

2. Ознайомитися з роботою використовуваних функцій, переглянувши вихідний код, а також коментарі у вихідних файлах бібліотеки і в режимі налагодження.

3. Створити новий проект в середовищі розробки для виконання індивідуального завдання, отриманого від викладача.

4. Реалізувати необхідну функціональність.

5. Запрограмувати плату і продемонструвати роботу програми.

Індивідуальні завдання

1. Налаштувати на відлагоджувальній платі один з модулів в режим прийому даних і прийняти дані від іншого пристрою з відображенням на семисегментному індикаторі.

2. Продемонструвати роботу в режимі ведучого з підключенням декількох пристроїв: перший пристрій - інший модуль SPI на платі, а другий - будь-який інший пристрій, який може виводити отримані дані на підключений індикатор.

3. Організувати передачу даних декільком підлеглим пристроям через певний інтервал часу. Підключити два пристрої до одного модуля SPI на

налагоджувальній платі і передавати дані поперемінно з інтервалом приблизно в 1 секунду з відображенням прийнятих даних.

4. Передача даних іншому модулю SPI на налагоджувальній платі (значення від 0x00 до 0x0F) з відображенням двійкового значення на світлодіодах на платі.

5. Одночасна передача даних декільком підлеглим пристроям, які відображають отримане значення на семисегментному індикаторі і в двійковому коді з використанням світлодіодів.

3.7 Лабораторна робота № 7.
Робота з DMA

Мета роботи: дослідити можливості використання DMA, основні характеристики DMA, взаємодія з іншими пристроями в складі мікроконтролера.

Обладнання та програмне забезпечення: налагоджувальна плата STM32F4 Discovery, мультиметр, середовище розробки STM32CubeIde, інструменти побудови проекту GNU Toolchain для ARM Embedded Processors, бібліотека CMSIS.

Теоретичні відомості

Прямий доступ до пам'яті (DMA) - Механізм, використовуваний в контролерах ARM для переміщення даних між пам'яттю і периферією без участі процесора. Ключовим моментом є те, що при використанні DMA на переміщення даних не використовуються ресурси процесора, що може бути особливо критичним при створенні додатків, що працюють з великою кількістю даних і активно використовують периферію. Робота DMA забезпечується окремим контролером, який виконує певні дії по команді процесора.

Розглянутий контролер має у своєму розпорядженні відразу два контролери DMA, які забезпечують у цілому 16 потоків (по 8 потоків на кожен контролер), кожен з яких призначений для управління запитами до пам'яті від одного або декількох пристроїв. Кожен потік може забезпечувати до 8 каналів (запитів). Кожен контролер DMA має пристрій вирішення конфліктів для обробки запитів відповідно до їх пріоритету.

Контролер DMA дозволяє проводити запис даних у трьох напрямках:

- від периферії в пам'ять;
- з пам'яті до периферії;
- з пам'яті в пам'ять.

Передача ведеться або в режимі безпосередньої передачі, або в режимі черги. Підтримується різний розмір даних для передачі, причому розмір даних для приймача і джерела може бути неоднаковим. У такому випадку DMA визначає дану ситуацію і виконує необхідні дії для оптимізації передачі, однак ця можливість підтримується тільки в режимі черги.

Крім того, дуже корисною може виявитися функція циклічної передачі, при використанні якої передача даних починається з початкової адреси знову після передачі останньої одиниці даних джерела.

Програміст може задавати пріоритети для запиту кожного конкретного потоку або пріоритет буде визначатися на основі значень за замовчуванням.

Приклад програми

Розглянемо приклад програми, яка використовує DMA для передачі даних з пам'яті в ЦАП. Зміна рівня сигналу на виході ЦАП відбувається циклічно з частотою зміни в 1с. Зміни відбуваються на основі значення, яке зчитується з пам'яті за сигналом переповнення таймера. У результаті цього ЦАП відправляє запит до DMA і отримує дані, які записуються в регістр даних, що призводить до зміни рівня сигналу. Схема взаємодії виглядає наступним чином (рис. 3.7).

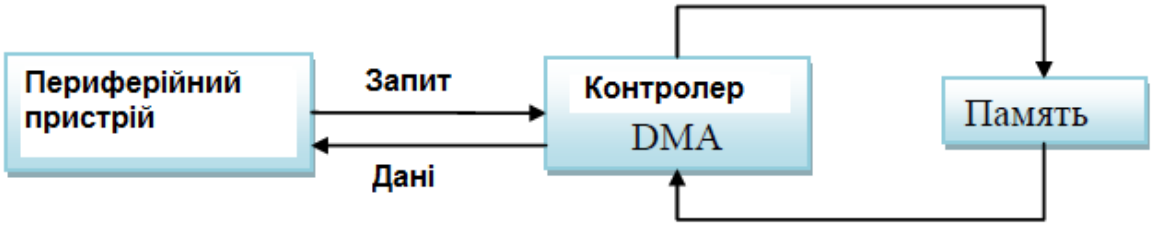

Рис. 3.7. Схема використання DMA

У самій програмі слід звернути увагу на велику кількість параметрів, необхідних для налаштування DMA в порівнянні з іншими пристроями. З цим пов'язана і складність використання DMA, оскільки необхідно враховувати велику кількість можливих налаштувань. Тим не менш, багато з них досить прості (напрям передачі, значення адрес), тому вивчення DMA можна зіставити за складністю з іншими пристроями, у чому дуже допомагає бібліотека Standard Peripheral Library.

```
#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_dac.h>

uint8_t levels[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77};

void init(void);
void init_gpio(void);
void init_timer(void);
void init_dac(void);
void init_dma(void);

int main(void)
{
      init();
      while(1)
      {
      }
}
```

```
void init(void) {
	init_gpio();
	init_timer();
	init_dac();
	init_dma();
}

void init_gpio(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	gpio_init.GPIO_Pin = GPIO_Pin_4;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOA, &gpio_init);
}

void init_timer(void) {
	TIM_TimeBaseInitTypeDef tim_init;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_Period = 16000 - 1;
	tim_init.TIM_Prescaler = 1000 - 1;
	TIM_TimeBaseInit(TIM6, &tim_init);

	TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);

	TIM_Cmd(TIM6, ENABLE);
}

void init_dac(void) {
	DAC_InitTypeDef dac_init;
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
	DAC_StructInit(&dac_init);
	dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
	dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
	dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
	DAC_Init(DAC_Channel_1, &dac_init);
}

void init_dma(void) {
	DMA_InitTypeDef dma_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
	DMA_DeInit(DMA1_Stream5);
	dma_init.DMA_Channel = DMA_Channel_7;
	dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
	dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
	dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
	dma_init.DMA_BufferSize = 8;
	dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
	dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
```

```
        dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
        dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
        dma_init.DMA_Mode = DMA_Mode_Circular;
        dma_init.DMA_Priority = DMA_Priority_High;
        dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
        dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
        dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
        dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
        DMA_Init(DMA1_Stream5, &dma_init);
        DMA_Cmd(DMA1_Stream5, ENABLE);

        DAC_Cmd(DAC_Channel_1, ENABLE);
        DAC_DMACmd(DAC_Channel_1, ENABLE);
}
```

Після компіляції програми-прикладу і прошивки налагоджувальної плати слід спостерігати за напругою на виході PA4 за допомогою мультиметра. Прилад повинен показати ступеневу зміну напруги від 0В до певного значення, після чого цикл повторюється. Крок збільшення однаковий для всього циклу, що можна побачити по масиву значень для запису в регістр даних.

Хід роботи

1. Перевірити роботу програми-прикладу, скомпілювавши її в одному із середовищ розробки і виконавши прошивку.
2. Ознайомитися з документацією по DMA для мікроконтролера на налагоджувальній платі.
3. Спробувати змінити певні параметри в програмі-прикладі.
4. Організувати взаємодію DMA і іншого периферійного пристрою відповідно до індивідуального завдання.

Індивідуальні завдання

1. Реалізувати асинхронну передачу даних через USART, зчитуючи дані через певні проміжки часу за допомогою DMA.
2. Організувати прийом даних через USART із записом в пам'ять через DMA. Кількість переданих даних відомо заздалегідь і дорівнює розміру масиву.
3. Продемонструвати передачу даних через SPI з пам'яті в циклічному режимі.
4. Продемонструвати прийом даних через SPI із записом в пам'ять. Режим запису - циклічний .
5. Реалізувати запис в пам'ять значень, отриманих за допомогою роботи АЦП через певні проміжки часу. Після запису перших 50 значень починається новий цикл запису.

3.8 Лабораторна робота № 8.
Генерація сигналів і управління їх характеристиками

Мета роботи: вивчити можливості генерації сигналів і способи управління їх характеристиками з використанням налагоджувальної плати та відображенням інформації.

Обладнання та програмне забезпечення: налагоджувальна плата STM32F4 Discovery, символьний LCD-дисплей, 3 тактові кнопки і резистори для їх підключення, середовище розробки для STM32 STM32CubeIde.

Приклад програми

В якості прикладу розглянемо програму, яка дозволяє на виході отримати сигнал прямокутної форми в заданому діапазоні частот і з заданим коефіцієнтом заповнення. Дана програма є свого роду закріпленням пройденого до цього матеріалу і охоплює багато вже пройдених моментів. Також її можна розглядати як продовження роботи з таймерами в режимі ШІМ. Відповідно користувачеві необхідно надати можливість для регулювання зазначених характеристик. Для прикладу взято такі значення:

- вихідна частота - 3-4 кГц;
- коефіцієнт заповнення ± 50 % від початкового значення, яким вважаємо значення в 0,5.

Для генерації сигналів прямокутної якнайкраще підходить включення таймера в режимі ШІМ для одного з вихідних каналів. Необхідно лише розрахувати значення для налаштування самого таймера при вказаній тактовій частоті пристрої (вони будуть різними для 16 МГц і 168 МГц) і включити тактування та потрібний режим роботи для інших пристроїв у складі контролера.

Відповідно до вказаних значень проведемо розрахунок для налаштування таймера (розглядається випадок з внутрішньої частотою 16 МГц). Оскільки більш детально цей процес описаний в одній з попередніх робіт, то далі наведено лише результати обчислень. Для того, щоб отримати на виході частоту в діапазоні 3-4кГц, встановлюємо значення переддільника рівне 10, тоді діапазон зміни значень в регістрі автозавантаження буде змінюватися від 533 до 400. Саме ці значення використовуються як граничні.

Коефіцієнт заповнення не потребує спеціальних розрахунків, тому лише скажемо, що це значення зберігається в програмі у вигляді дійсного числа, яке користувач може змінювати і зазначені зміни записуються в регістр порівняння для обраного вихідного каналу.

В якості вирішення для надання користувачеві можливості регулювання параметрів вихідного сигналу пропонується також підключити символьний LCD-дисплей і три кнопки для виконання регулювання. Функціональне

призначення кнопок може змінюватися відповідно до дій користувача. У той же час на екрані слід відображати інформацію про поточні зміни (зміна пунктів меню, значень характеристик), які виконує користувач.

Перейдемо до ознайомлення з кодом проекту. Функціональна частина оголошена і реалізована в файлах init.h і init.c. Далі наводиться лістинг файлу init.c.

```
#include "init.h"
#include "hd44780lib.h"
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_exti.h>
#include <stm32f4xx_syscfg.h>
#include <stm32f4xx_tim.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <misc.h>

// ініціалізація портів, до який під’єднано дисплей
void init_gpio_lcd(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOE,
ENABLE);
	gpio_init.GPIO_Pin = DATA_PINS;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(DATA_PORT, &gpio_init);
	gpio_init.GPIO_Pin = CONTROL_PINS;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(CONTROL_PORT, &gpio_init);
}

// ініціалізація порту, до якого під’єднанні кнопки
void init_gpio_buttons(void) {
	GPIO_InitTypeDef gpio_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_IN;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Pin = BUTTON_PINS;
	GPIO_Init(GPIOB, &gpio_init);
}

// configures buttons inputs on board as interrupt source
void configure_buttons(void) {
	EXTI_InitTypeDef exti_init;
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
```

```
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource1);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource2);
	exti_init.EXTI_LineCmd = ENABLE;
	exti_init.EXTI_Line = EXTI_Line0 | EXTI_Line1 | EXTI_Line2;
	exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
	exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
	EXTI_Init(&exti_init);
}

// функція дозволу вказаного переривання
void enable_interrupt(IRQn_Type irq) {
	NVIC_InitTypeDef nvic_init;
	nvic_init.NVIC_IRQChannel = irq;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelSubPriority = 1;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_Init(&nvic_init);
}

// дозволяємо преривання по натисненню кнопки
void configure_interrupts(void) {
	enable_interrupt(EXTI0_IRQn);
	enable_interrupt(EXTI1_IRQn);
	enable_interrupt(EXTI2_IRQn);
}

void configure_delay_timer(void) {
	TIM_TimeBaseInitTypeDef tim_init;
	tim_init.TIM_CounterMode = TIM_CounterMode_Up;
	tim_init.TIM_Prescaler = 1600 - 1;//налаштовуєм передільник таймера
для затримок
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
	TIM_TimeBaseInit(DELAY_TIMER, &tim_init);
}

// налаштування таймера для генерації ШІМ
void configure_pwm_timer(void) {
	GPIO_InitTypeDef gpio_init;
	TIM_TimeBaseInitTypeDef tim_init;
	TIM_OCInitTypeDef oc_init;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
	gpio_init.GPIO_Pin = GPIO_Pin_0;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOA, &gpio_init);

	GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_TIM2);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	tim_init.TIM_Prescaler = 10 - 1;
```

```
            tim_init.TIM_Period = frequency_value - 1;
            tim_init.TIM_CounterMode = TIM_CounterMode_Up;
            tim_init.TIM_ClockDivision = 0;
            tim_init.TIM_RepetitionCounter = 0;
            TIM_TimeBaseInit(PWM_TIMER, &tim_init);

            oc_init.TIM_OCMode = TIM_OCMode_PWM1;
            oc_init.TIM_Pulse = frequency_value / 2;
            oc_init.TIM_OutputState = TIM_OutputState_Enable;
            TIM_OC1Init(PWM_TIMER, &oc_init);

            TIM_Cmd(PWM_TIMER, ENABLE);
      }

      // заповнюмо меню
      void fill_menu(void) {
            main_option_index = 0;
            main_options[0] = create_menu_item("1.Change frequency",
set_active_freq_menu);
            main_options[1] = create_menu_item("2. Change fill",
set_active_fill_menu);

            main_menu.items[0] = create_menu_item("", down_menu);
            main_menu.items[1] = main_options[main_option_index];
            main_menu.items[2] = create_menu_item("", up_menu);

            freq_menu.items[0] = create_menu_item("", decrement_freq);
            freq_menu.items[1] = create_menu_item("", set_active_main_menu);
            freq_menu.items[2] = create_menu_item("", increment_freq);

            filling_menu.items[0] = create_menu_item("", increment_fill);
            filling_menu.items[1] = create_menu_item("", set_active_main_menu);
            filling_menu.items[2] = create_menu_item("", decrement_fill);
            set_active_menu(&main_menu);
      }

      void set_active_menu(struct menu * menu) {
            active_menu = menu;
      }

      void set_active_freq_menu(void) {
            active_menu = &freq_menu;
            show_freq_on_screen();
      }

      void set_active_fill_menu(void) {
            active_menu = &filling_menu;
            show_fill_on_screen();
      }

      void set_active_main_menu(void) {
            active_menu = &main_menu;
```

```
            show_text_on_screen(&main_menu.items[1]);
      }

      // реалізація функції затримки
      void delay(int times) {
            DELAY_TIMER->CNT = 0;
            DELAY_TIMER->ARR = times;
            TIM_Cmd(DELAY_TIMER, ENABLE); // запуск таймера
            while(TIM_GetFlagStatus(DELAY_TIMER, TIM_FLAG_Update) == RESET); //
очікування
            TIM_ClearFlag(DELAY_TIMER, TIM_FLAG_Update);
            TIM_Cmd(DELAY_TIMER, DISABLE); // зупинка таймера
      }

      struct menu_item create_menu_item(char * text, action act) {
            struct menu_item item;
            strcpy(item.menu_text, text);
            item.action = act;
            return item;
      }

      void show_text_on_screen(struct menu_item * item) {
           lcd_clear();
           write_string(item->menu_text);
      }

      // збільшення значення змінною для зміни частоти
      void increment_freq(void) {
           if (frequency_value == MAX_FREQ_VALUE) return;
           frequency_value++;
           TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
           TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
           show_freq_on_screen();
      }

      // зменшення значення змінної для зміни частоти
      void decrement_freq(void) {
           if (frequency_value == MIN_FREQ_VALUE) return;
           frequency_value--;
           TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
           TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
           show_freq_on_screen();
      }

      void increment_fill(void) {
           if (MAX_FILL_VALUE - fill_value < 0.01) return;
           fill_value += 0.01;
           TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
           show_fill_on_screen();
      }

      void decrement_fill(void) {
```

```
    if (fill_value - MIN_FILL_VALUE < 0.01) return;
    fill_value -= 0.01;
    TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
    show_fill_on_screen();
}
void change_menu_item(struct menu_item * item, char * text, action act) {
    strcpy(item->menu_text, text);
    item->action = act;
}

void chamge_menu_item(struct menu_item * cur, struct menu_item * next) {
     cur = next;
}

void up_menu(void) {
    if (main_option_index == 1) {
    } else {
          main_option_index++;
          main_menu.items[1] = main_options[main_option_index];
          show_text_on_screen(&main_menu.items[1]);
     }
}

void down_menu(void) {
    if (main_option_index == 0) {
    } else {
          main_option_index--;
          main_menu.items[1] = main_options[main_option_index];
          show_text_on_screen(&main_menu.items[1]);
     }
}

void show_freq_on_screen(void) {
    char text[20];
    double freq = (double)16000000 / (frequency_value * 10) / 1000;
    gcvt(freq, 4, text);
    strcat(text, " kHz");
    lcd_clear();
    write_string(text);
}

void show_fill_on_screen(void) {
    char text[20];
    gcvt(fill_value, 4, text);
    strcat(text, " %");
    lcd_clear();
    write_string(text);
}

// ініціалізація всіх компонентів
void init_all(void) {
    frequency_value = 533;
```

```
    fill_value = 0.5;
    init_gpio_buttons();
    init_gpio_lcd();
    configure_buttons();
    configure_interrupts();
    configure_delay_timer();
    configure_pwm_timer();
    fill_menu();
    lcd_init();
    show_text_on_screen(&main_menu.items[1]);
}
```

У заголовному файлі знаходяться оголошення функцій, а також використовуваних змінних і констант.

```
#ifndef INIT_H
#define INIT_H

#include <stm32f4xx.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>

#define BUTTON_PINS GPIO_Pin_0 | \
                                    GPIO_Pin_1 | \
                                    GPIO_Pin_2
#define DELAY_TIMER TIM7
#define PWM_TIMER TIM2
#define MAX_FREQ_VALUE 533
#define MIN_FREQ_VALUE 400
#define MAX_FILL_VALUE 0.75
#define MIN_FILL_VALUE 0.25

typedef void (*action)(void);

/*
* Структура для представлення пункту меню
*/

struct menu_item {
      action action; // действие
      char menu_text[20]; // текст меню
};

struct menu {
      struct menu_item items[3]; // пункти меню
};

int frequency_value; // зміння для задання частоти, має значення 400...533
double fill_value; // змінна для задання коэфіцієнту заповнення
struct menu * active_menu; // поточне меню
struct menu_item main_options[3]; // пункти головного меню
struct menu main_menu;
struct menu freq_menu;
```

```
struct menu filling_menu;
int main_option_index; // індекс поточного пункту в головному меню
void init_gpio_lcd(void);
void init_gpio_buttons(void);

void configure_buttons(void);
void configure_interrupts(void);
void configure_delay_timer(void);
void configure_pwm_timer(void);

void init_all(void);

struct menu_item create_menu_item(char * text, action act);
void show_text_on_screen(struct menu_item * item);
void show_freq_on_screen(void);
void show_fill_on_screen(void);
void fill_menu(void);
void set_active_menu(struct menu * menu);
void set_active_freq_menu(void);
void set_active_fill_menu(void);
void set_active_main_menu(void);
void empty_action(void);
void change_menu_item(struct menu_item * item, char * text, action act);
void chamge_menu_item(struct menu_item * cur, struct menu_item * next);
void up_menu(void);
void down_menu(void);

void increment_freq(void);
void decrement_freq(void);
void increment_fill(void);
void decrement_fill(void);

#endif // INIT_H
```

У файлі з основною функцією викликається функція ініціалізації всіх використовуваних пристроїв і описані обробники переривань для натискань кнопок.

```
#include <stm32f4xx_exti.h>
#include <string.h>
#include "hd44780lib.h"
#include "init.h"

void EXTI0_IRQHandler(void) {
      EXTI_ClearITPendingBit(EXTI_Line0);
      if (active_menu != NULL)
            delay(100);
            active_menu->items[0].action();
}

void EXTI1_IRQHandler(void) {
      EXTI_ClearITPendingBit(EXTI_Line1);
```

```
	if (active_menu != NULL)
		delay(100);
		active_menu->items[1].action();
}

void EXTI2_IRQHandler(void) {
	EXTI_ClearITPendingBit(EXTI_Line2);
	if (active_menu != NULL)
		delay(100);
		active_menu->items[2].action();
}

int main(void)
{
		init_all();
	while(1)
	{
	}
}
```

Опис установки

У підключенні беруть участь 3 порти вводу/виводу (виводу порту, на якому генерується сигнал, підключається вимірювальне обладнання). Кнопки підключаються до виводів 0-2 порту B. Вихідний сигнал можна спостерігати на виході 0 порту A.

Схема підключення компонентів показана на рис. 3.8.

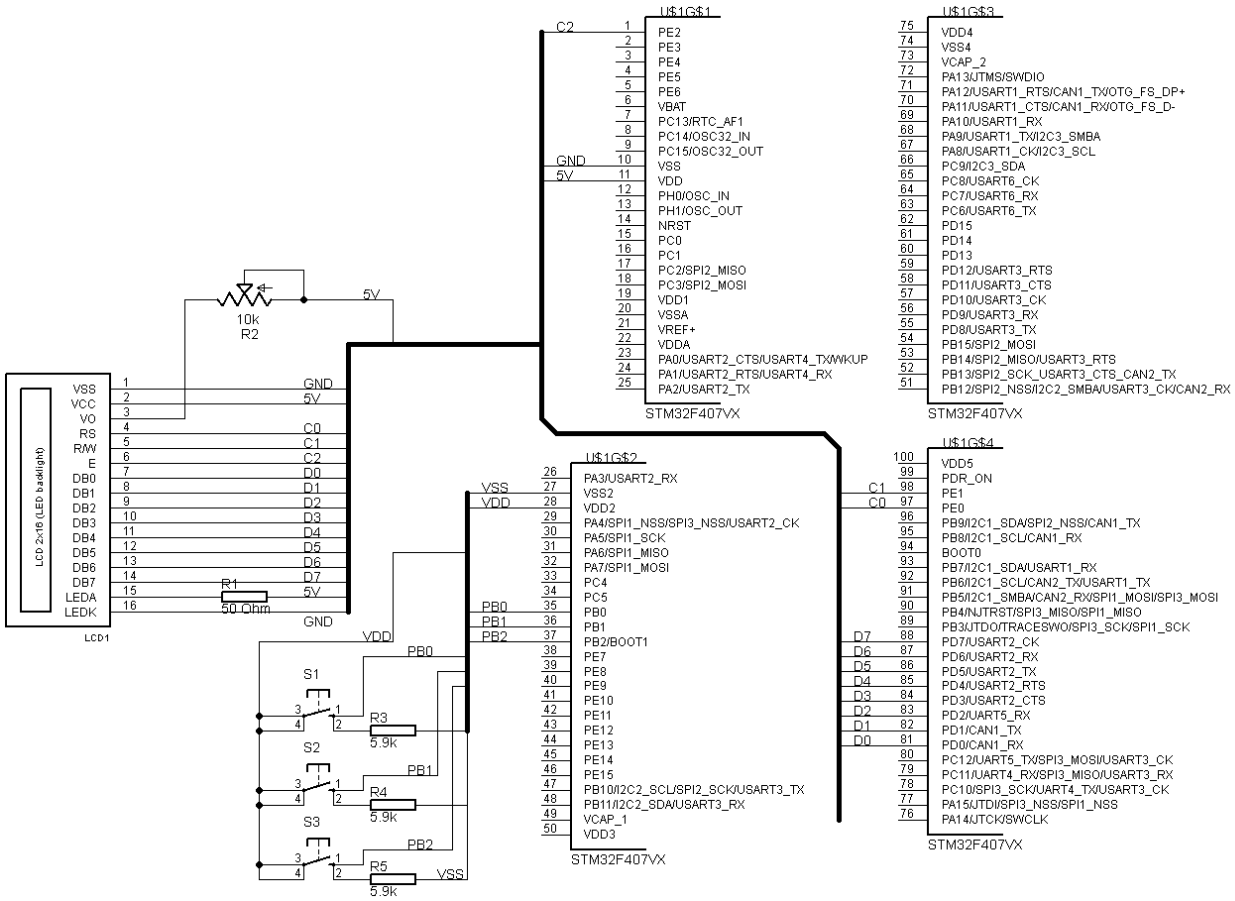

Рис. 3.8 . Схема підключення компонентів

Для управління варіантами меню, а також зміною характеристик використовуються підключені до виходів 0-2 порту B тактові кнопки. У даному прикладі використані 4- контактні тактові кнопки TS- 06V (рис. 3.9), однак можна використовувати й інші кнопки, які вдасться знайти.

Рис. 3.9. Зовнішній вигляд тактової кнопки

Перед тим, як виконати підключення, слід вивчити за допомогою мультиметра в режимі вимірювання опору, які саме контакти замикаються при натисканні, а які залишаються з'єднаними завжди. Крім того, для підтяжки виходу до землі використовуються резистори номіналом 5,9 кОм.

Для розміщення описаного вище з'єднання використовується макетна плата. Усі виводи на платі (3 - для кнопок і 2 - для живлення і землі) для

зручності підключені до штирьового роз'єму. Підключення до налагоджувальної плати виконується за допомогою перемичок.

Індивідуальні завдання

1. Максимальна частота роботи контролера на платі складає 168 МГц. Використовуючи інструкцію за заданням частот, виконати розрахунок і запустити розглянуті приклади з дотриманням значень параметрів частоти і коефіцієнта заповнення.

2. За аналогією з наведеним прикладом напишіть програму, яка дозволяє змінювати частоту прямокутного сигналу в діапазоні від 1 до 100 Гц і регулювати коефіцієнт заповнення у всьому діапазоні. Роботу продемонструвати на частоті 1 Гц, змінюючи значення коефіцієнта.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Noviello C. Mastering STM32. Rel.0.21 // Leanpub, 2019. — 819 p.
2. Watanabe K. Introduction to STM32 ARM Microcontroller with STM HAL-Library & SW4STM32 (+ sources code) // Amazon Digital Services LLC, 2019. — 99 p
3. Peter Marwedel Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things // Germany, Dortmund.- Springer.-2021.-455p.
4. Jonathan W. Valvano Embedded Systems: Introduction to Robotics // USA, Traverse City, Michigan.-Independently published.-2019.-499p.
5. Kamila Meyer Real Time Operating System And Its Basics: A Simple Guide To Real-Time Operating System For Absolute Beginners | Fundamentals Of Real-Time Programming For Embedded Systems For Non-Experts // USA, Traverse City, Michigan.-Independently published.-2022.-367p.
6. Philip Koopman Better Embedded System Software // USA, Traverse City, Michigan.-Independently published.-2021.-384p.
7. Randall, Hyde Book Of I2C, The: A Guide for Adventurers // USA, San Francisco.- No Starch Press.-2022.-448p.
8. Dr. Jim Cooling Real-time Operating Systems Book 1: The Theory // USA, Traverse City, Michigan.-Independently published.-2019.-325p.
9. Warren Gay Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC // USA, New York.-Apress Media, LLC.-2018.-430p.
10. Pakdel Majid Advanced Programming with STM32 Microcontrollers: Master the software tools behind the STM32 microcontroller // Netherlands, Susteren.- Elektor International Media BV.- 2020.-216p.

Навчально-методичне видання

ЗАСТАВНИЙ Олег Михайлович

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з курсу:

«ПРОЕКТУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ»

для студентів ступеня вищої освіти «бакалавр»
спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології»

Підписано до друку 01.02.2023 р.
Формат 60х84/16. Папір офсетний.
Друк офсетний. Зам. № 23-229
Умов.-друк. арк. 3,9. Обл.-вид. арк. 4,1.
Тираж 30 прим.

Віддруковано ФО-П Шпак В. Б.
Свідоцтво про державну реєстрацію В02 № 924434 від 11.12.2006 р.
м. Тернопіль, бульвар Просвіти, 6/4. тел. 097 299 38 99.
E-mail: tooums@ukr.net

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 7599 від 10.02.2022 р.