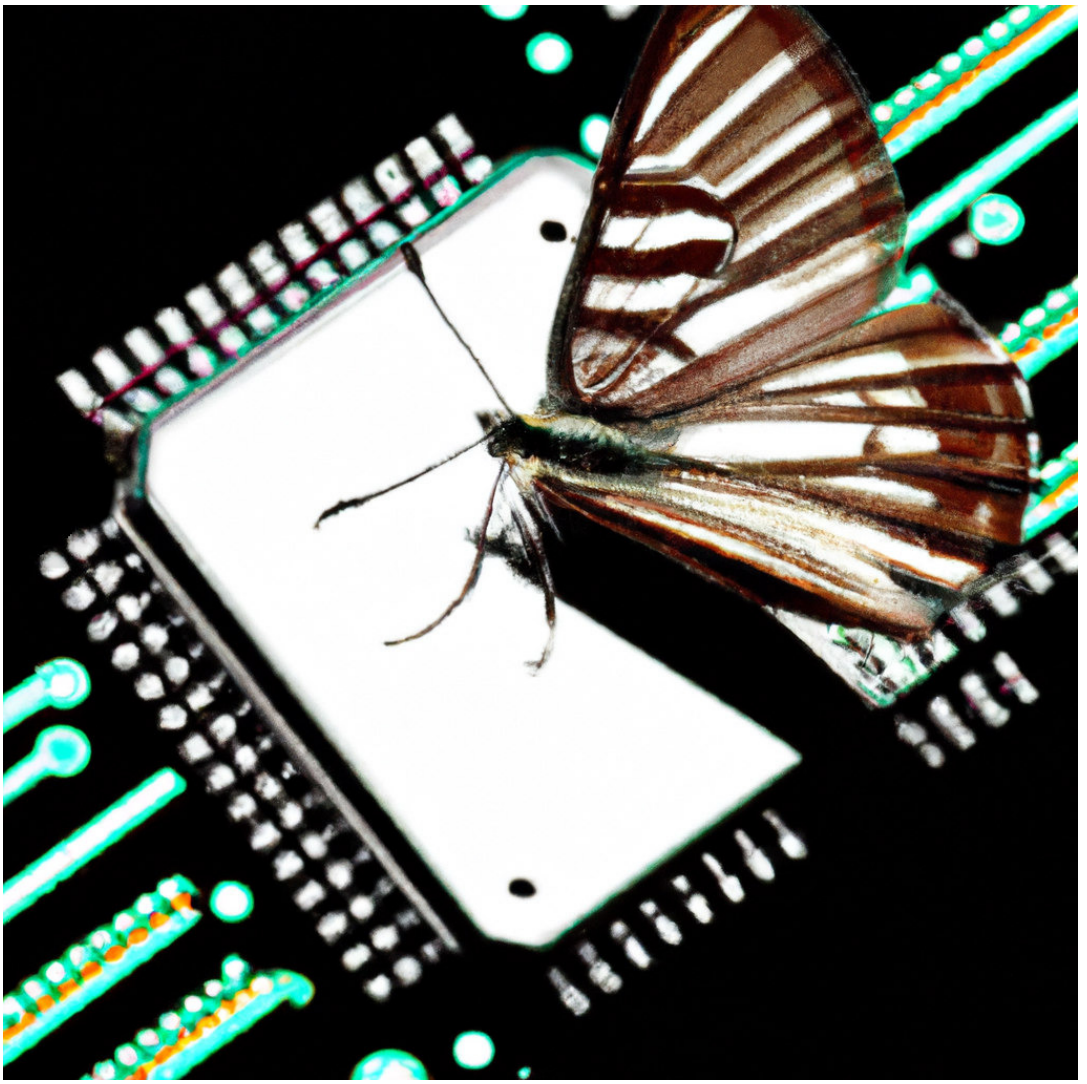
ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ STM32 У STM32CUBEIDE



ПРАКТИКУМ

Міністерство освіти і науки України
Національний університет водного господарства та природокористування

Д. Т. Реут

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ STM32 У STM32CUBEIDE. ПРАКТИКУМ

Навчальний посібник

Рівне – 2023

УДК 004.4'2
Р44

Рецензенти:

Матіко Ф.Д., доктор технічних наук, професор, завідувач кафедри автоматизації та комп'ютерно-інтегрованих технологій Національного університету «Львівська політехніка»

Сафоник А.П., доктор технічних наук, професор, в. о. директора навчально-наукового інституту енергетики, автоматики та водного господарства Національного університету водного господарства та природокористування

Рекомендовано Вченою радою Національного університету водного господарства та природокористування.
Протокол № 10 від 27 жовтня 2023 р.

Реут Д.Т.

Р44 Програмування мікроконтролерів STM32 у STM32CubeIDE. Практикум : навч. посіб. [Електронне видання]. – Рівне : НУВГП, 2023. – 120 с.

ISBN 978-966-327-573-4

У навчальному посібнику охоплено практичну частину модуля 2 навчальної дисципліни «Мікропроцесорні системи та програмування мікропроцесорних засобів» і розглянуто програмування мікроконтролерів серії STM32F0 на прикладі мікроконтролера STMicroelectronics STM32F072C8T6 у середовищі розробки STM32CubeIDE. Представлені лабораторні та практичні роботи, направлені на формування вмінь використовувати порти вводу-виводу загального призначення, аналогово-цифровий перетворювач, таймери, контролер прямого доступу до пам'яті та модуль USART.

Посібник рекомендовано для студентів, які вивчають дисципліну «Мікропроцесорні системи та програмування мікропроцесорних засобів» спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» та 141 «Електроенергетика, електротехніка та електромеханіка».

УДК 004.4'2

ISBN 978-966-327-573-4

ЗМІСТ

ВСТУП

Найбільш розповсюдженими мікропроцесорами у світі є мікроконтролери. Обсяги їх виробництва на порядок перевищують обсяги виробництва універсальних мікропроцесорів. Мікроконтролери можна знайти в бездротових навушниках і в ABS автомобіля, в кавомашині та в контролері системи автоматизації реактора на фабриці, в POS-терміналі та в роботі-пилососі, в пральній машині та в контролері закрилка літака, в розумній розетці та в апараті штучної вентиляції легень.

Вони виконують роль центрального цифрового керуючого елемента, збираючи дані від датчиків, обмінюючись даними з іншими вузлами мережі, формуючи сигнали керування приводами.

На ринку мікроконтролерів спостерігається тенденція переходу на 32-бітні мікроконтролери з 8- і 16-бітних. Значну частину ринку 32-бітних мікроконтролерів займають мікроконтролери родини STM32 від європейського виробника STMicroelectronics. Серія STM32F0 охоплює ціновий діапазон 8- і 16-бітних мікроконтролерів, тому часто використовується в нових електронних пристроях, чутливих до вартості компонентів. Можливості периферії й продуктивність мікроконтролерів родини STM32F0 часто кращі, ніж відповідних за ціною 8- і 16-бітних мікроконтролерів.

Розробка програм для мікроконтролерів цієї серії значно спрощується при використанні бібліотек рівня апаратних абстракцій HAL і середовища розробки STMicroelectronics STM32CubeIDE. Саме їх використання передбачено в цій книзі.

Практикум сфокусований на застосуванні мікроконтролерів у практичних задачах на прикладі STMicroelectronics STM32F072C8T6. Щоб пришвидшити складання апаратної частини на практичних і лабораторних роботах, мікроконтролер розпаяний разом з іншими електронними компонентами на налагоджувальній платі, проте для самостійного виконання завдань можуть використовуватись й окремі компоненти, що підключаються на макетній платі, та інші налагоджувальні плати, наприклад STMicroelectronics NUCLEO-F072RB або 32F072BDISCOVERY. Зображення мідного провідного шару для верхньої та нижньої сторони плати й список компонентів для виготовлення плати власноруч наведені в додатках А, Б, В.

Для виконання практичної частини бажаний досвід програмування 8-бітних мікроконтролерів мовою C або C++ й базові знання електроніки. В протилежному випадку рекомендовано спершу навчитись розробляти програми для 8-бітного мікроконтролера,

наприклад, Microchip ATmega328P на платі Arduino Uno R3, і складати схеми з останньою.

У практикумі розглянуто встановлення і налаштування середовища розробки STM32CubeIDE, використання портів вводу-виводу загального призначення, таймерів у режимі ШІМ, контролера прямого доступу до пам’яті, аналового-цифрового перетворювача, універсального синхронно-асинхронного прийомопередавача, модуля USB. Розробка програм для лабораторних та практичних робіт супроводжується детальними поясненнями щодо причин використання того чи іншого елемента.

Практикум призначений для студентів, які вивчають дисципліну «Мікропроцесорні системи та програмування мікропроцесорних засобів» спеціальностей 151 «Автоматизація та комп’ютерно-інтегровані технології», 141 «Електроенергетика, електротехніка та електромеханіка».

1. ВСТАНОВЛЕННЯ СЕРЕДОВИЩА РОЗРОБКИ STM32CUBEIDE

STMicroelectronics STM32CubeIDE є крос-платформеним середовищем розробки програм для мікроконтролерів та мікропроцесорів STM32, яке окрім звичних функцій компіляції та налагодження програми містить також конфігуратор периферії мікроконтролера з функцією генерування програмного коду.

STM32CubeIDE базується на середовищі програмування з відкритим вихідним кодом Eclipse CDT, компіляторі GCC ARM, налагоджувачі GDB та засобі генерування коду початкової ініціалізації STM32CubeMX. Останній дозволяє налаштувати потрібні модулі мікроконтролера в графічному режимі й згенерувати проект, що вже містить програмний код для початкового налаштування периферії, звільняючи розробника від потреби писати його вручну. Це дозволяє використовувати вже існуючі плагіни для Eclipse IDE.

Історично першим середовищем розробки для STM32 на базі Eclipse, GCC ARM та GDB є System Workbench for STM32 від Ac6. Багато статей з розробки програм для мікроконтролерів STM32 містять інструкції саме для нього. У випадку виконання практичної частини в цьому середовищі знадобиться окремо встановити генератор коду ініціалізації STM32CubeMX. Компанія STMicroelectronics позначає згадане середовище як NRND (Not Recommended for New Design), тобто не рекомендує його для нових розробок. Проект, розроблений в System Workbench for STM32, може автоматично імпортуватись в STM32CubeIDE.

STM32CubeIDE також включає стандартні та розширені функції налагодження, включаючи перегляд регістрів ядра процесора, пам’яті та периферійних регістрів, а також перегляд змінних у реальному часі.

STM32CubeIDE можна завантажити на сторінці https://www.st.com/en/development-tools/stm32cubeide.html. Далі слід дотримуватись інструкцій з підрозділів 1.1-1.3 залежно від операційної системи вашого комп’ютера.

1.1. Встановлення STM32CubeIDE в Windows

ZIP-архів з інсталятором для Windows, завантаженим з офіційного сайту, потрібно розпакувати з запустити інсталятор stm32cubeide_..._x86_64.exe від імені адміністратора. Погодитись з ліцензійною угодою, вибрати каталог для встановлення, після чого в додаткових компонентах для встановлення відмітити ST-LINK drivers для встановлення (рис. 1.1). Без драйвера завантаження й

налагодження програми в мікроконтролері програматором-налагоджувачем ST-link буде неможливе.

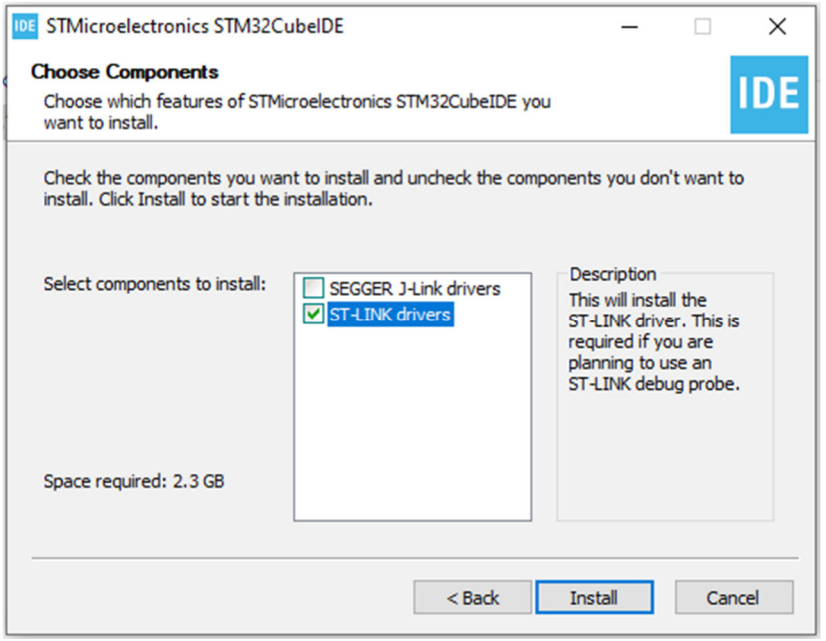


Рис. 1.1. Вибір драйвера ST-LINK для встановлення

Після цього запускаємо встановлення кнопкою «Install». Під час встановлення будуть встановлені драйвери, потрібні для підключення мікроконтролера STM32 до ПК через програматор-налагоджувач, підписані STMICROELECTRONICS (GRENOBLE 2) SAS, відповідно потрібно погодитись на встановлення драйверів від цього видавця (рис. 1.2).

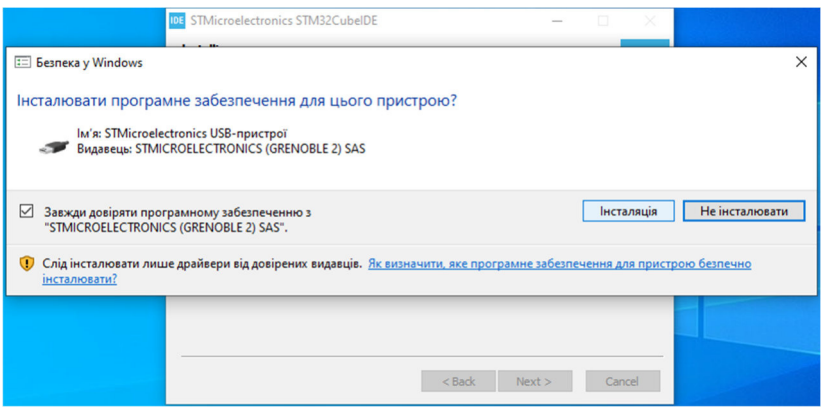


Рис. 1.2. Підтвердження встановлення драйверів для взаємодії з STM32

Після закінчення процесу інсталяції все готово для створення першого проекту.

1.2. Встановлення STM32CubeIDE в Ubuntu Linux

Для дистрибутивів GNU/Linux виконати встановлення можна з архіву, який доступний на офіційному сайті як STM32CubeIDE-Lnx (STM32CubeIDE Generic Linux Installer). Цей архів слід розпаковуємо unzip'ом або засобами GUI й отримуємо файл st-stm32cubeide_... _amd64.sh. Останній слід запустити як виконуваний файл в терміналі (рис. 1.3).

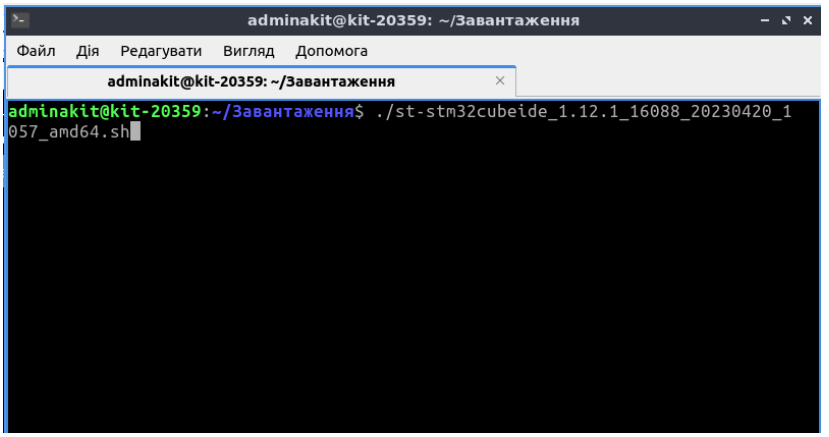


Рис. 1.3. Запуск STM32CubeIDE Generic Linux Installer

Погодитись з умовами ліценійної угоди STM32CubeIDE, прогорнувши її до кінця вниз натисканням клавіші Enter, натиснувши «y» і підтвердивши вибір натисканням Enter (рис. 1.4).

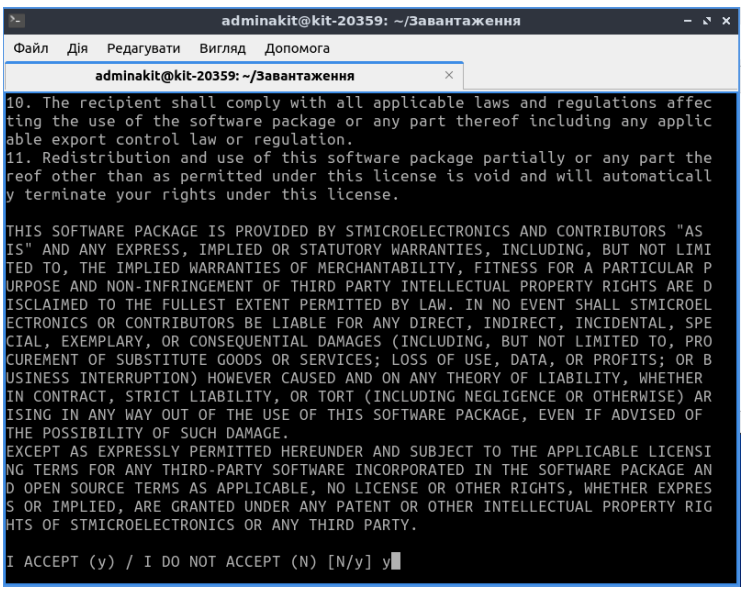


Рис. 1.4. Підтвердження згоди з умовами ліцензійної угоди STM32CubeIDE Generic Linux Installer

Наступним кроком потрібно підтвердити встановлення stlink-server і додавання правил udev. Без цього пункту завантаження та налагодження програми через ST-link буде неможливе. Далі вказуємо шлях до каталогу встановлення (рис. 1.5), настикаємо Enter і запускаємо власне встановлення. Під час встановлення у відповідь на запрошення програми sudo потрібно ввести пароль поточного користувача з правами адміністратора.

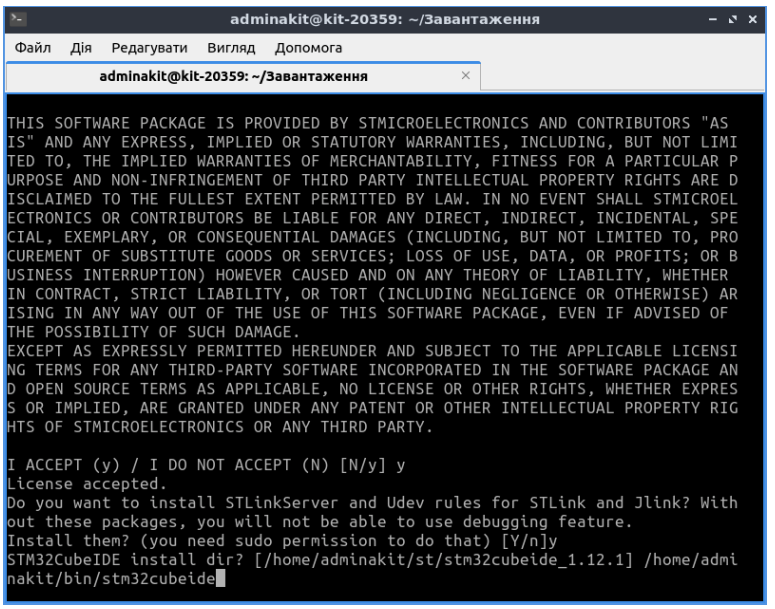


Рис. 1.5. Вказання шляху встановлення STM32CubeIDE

Після закінчення процесу інсталяції все готово для створення першого проекту.

Також середовище STM32CubeIDE може бути встановлене як набір deb-пакетів у операційній системі Ubuntu Linux або Debian. Для цього з офіційного сайту слід завантажувати STM32CubeIDE-DEB (STM32CubeIDE Debian Linux Installer) замість STM32CubeIDE-Lnx (STM32CubeIDE Generic Linux Installer). Завантажений zip-архів слід розпакувати й отриманому shell-скрипту дати права на виконання й запустити його від імені суперкористувача в терміналі командою *sudo ./st-stm32cubeide_..._amd64.deb_bundle.sh* . Аналогічно попередньому випадку погодитись з ліцензійними умовами та встановленням правил udev, після чого почнеться завантаження залежностей та встановлення пакетів stlink-server та stm32cubeide.

Після закінчення процесу інсталяції всіх deb-пакетів усе готово для створення першого проекту

1.3. Встановлення STM32CubeIDE в macOS

ZIP-архів з інсталятором для macOS, завантаженим з офіційного сайту, потрібно розпакувати й відкрити файл st-stm32cubeide_... _x86_64.dmg. Погодитись з ліцензійною угодою (рис. 1.6).

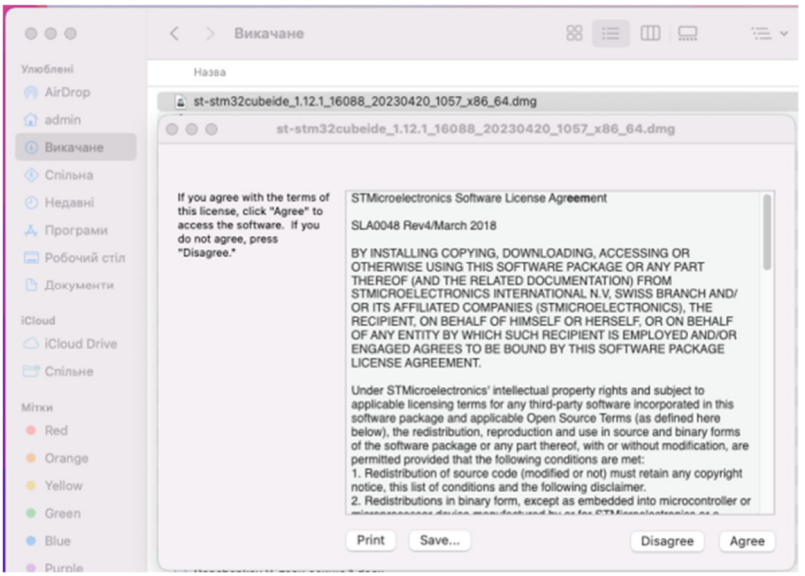


Рис. 1.6. Вікно ліцензійної угоди STM32CubeIDE для macOS

Після відображення вмісту dmg-пакету (рис. 1.7) першочергово слід встановити st-stlink….pkg, який містить драйвер програматора-налагоджувача ST-link та stlink-server.

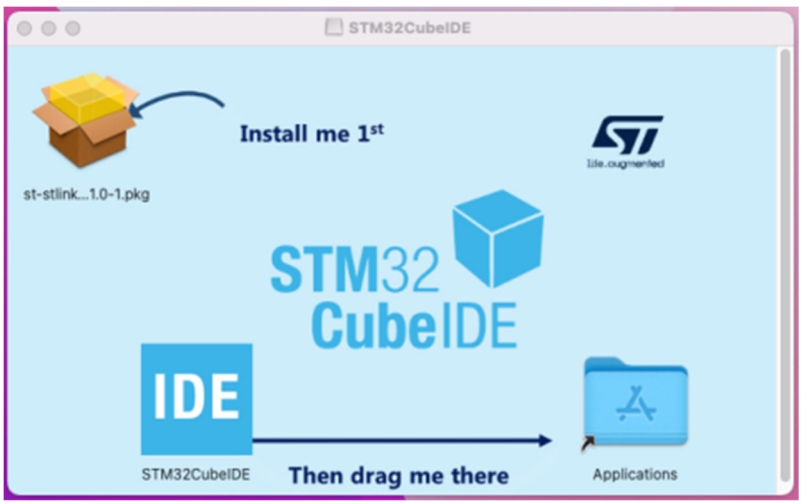


Рис. 1.7. Вміст dmg-пакету з STM32CubeIDE

Погодитись відкрити програму у випадку попередження macOS про неможливість перевірити розробника програми (рис. 1.8). Після

завершення встановлення st-stlink….pkg можна звичним способом перетягнути програму, що встановлюється (STM32CubeIDE), в теку зі встановленими програмами.

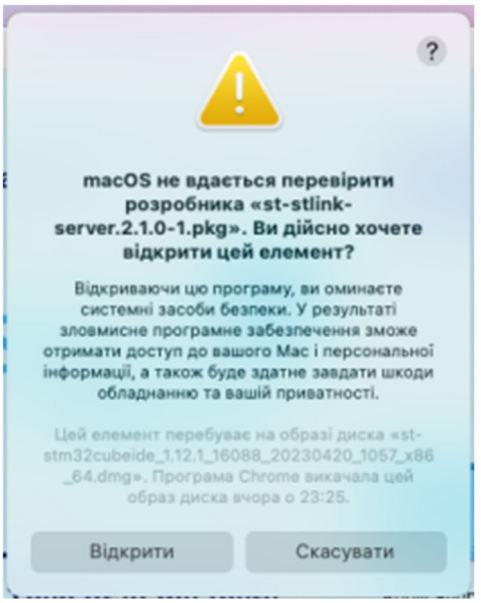


Рис. 1.8. Попередження macOS про неможливість перевірити розробника програми st-stlink-server

При спробі запуску STM32CubeIDE macOS буде видавати попередження про неможливість перевірити розробника програми й відмову запускати неперевірену програму. Щоб ігнорувати це попередження, у програмі Finder в папці «Програми» знайдіть програму STM32CubeIDE. Клацніть клавішу Control, потім — іконку програми та виберіть «Відкрити» з контекстного меню. У діалоговому вікні з попередженням натисніть кнопку «Відкрити», щоб запустити неперевірену програму. Ця програма збережеться як виняток у параметрах безпеки, і ви зможете надалі запускати її просто подвійним натисканням, як будь-яку авторизовану програму.

Все готово для створення першого проекту.

2. ПЕРШИЙ ПРОЕКТ В STM32CUBEIDE

Практикум сфокусовано на використання мікроконтролерів серії STM32F0 для вирішення практичних задач з використанням рівня апаратних абстракцій HAL. Архітектурні особливості мікроконтролерів STM32F0 на базі архітектури ARM Cortex-M0 приховані «під капотом» HAL [1], розробник при реалізації типових задач, як правило, не має потреби до них звертатись напряму (за вийнятком реалізації операцій, критичних до часу). Всі наступні проекти в практикумі використовуватимуть HAL.

2.1. Створення проекту й компіляція програми

Почнемо знайомство з «Hello, world!» у світі мікроконтролерів – блимання світлодіодом.

У STM32CubeIDE в головному меню обираємо File->New->STM32 Project. Фільтруємо список мікроконтролерів (рис. 2.1): вводимо в поле Commercial Part Number потрібну модель мікроконтролера (STM32F072C8T6). Якщо використовується плата з іншим мікроконтролером – замість STM32F072C8T6 вказуємо його модель.

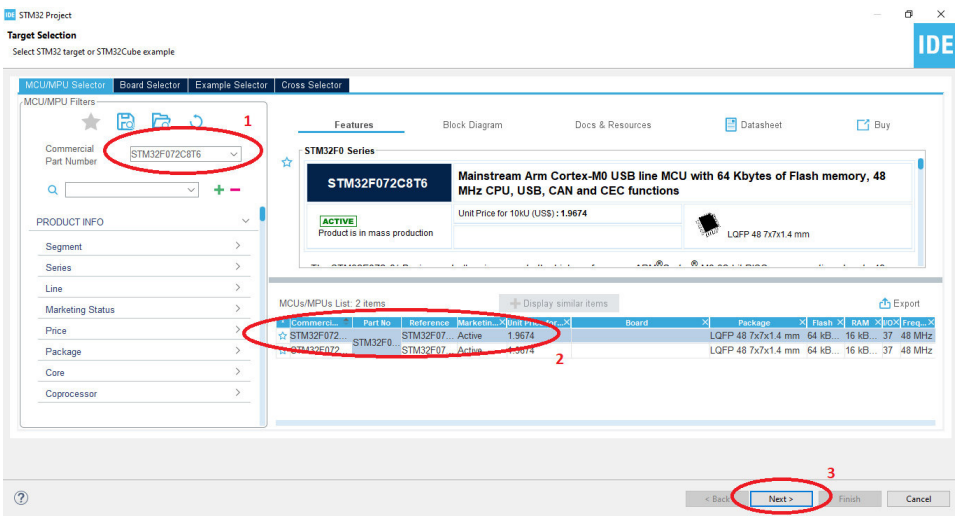


Рис. 2.1. Вибір мікроконтролера при створенні проекту

Задаємо назву проекту (напр, blink), цільову мову C, тип цілового файлу – виконуваний файл, тип проекту – STM32Cube. Підтримується також створення проекту мовою C++ та компіляція статичної бібліотеки замість виконуваного файлу, який можна прошити в мікроконтролер.

Після створення проекту відкриється файл blink.ioc, в якому можна в графічному режимі налаштувати периферію мікроконтролера. Він містить налаштування мікроконтролера й додаткових бібліотек, а також налаштування проекту для генерування програмного коду. Раніше ioc-файл створювався в окремій програмі STM32CubeMX, яка дозволяє згенерувати програмний код початкової ініціалізації для різних середовищ розробки й компіляторів. Компоненти STM32CubeMX інтегровані в середовище розробки STM32CubeIDE, відповідно при збереженні ioc-файла останнє запропонує згенерувати відповідний програмний код початкової ініціалізації в створеному проекті.

На вкладці Pinout & Configuration (рис. 2.2) оберемо режим роботи PA8 як цифровий вихід (GPIO_Output).

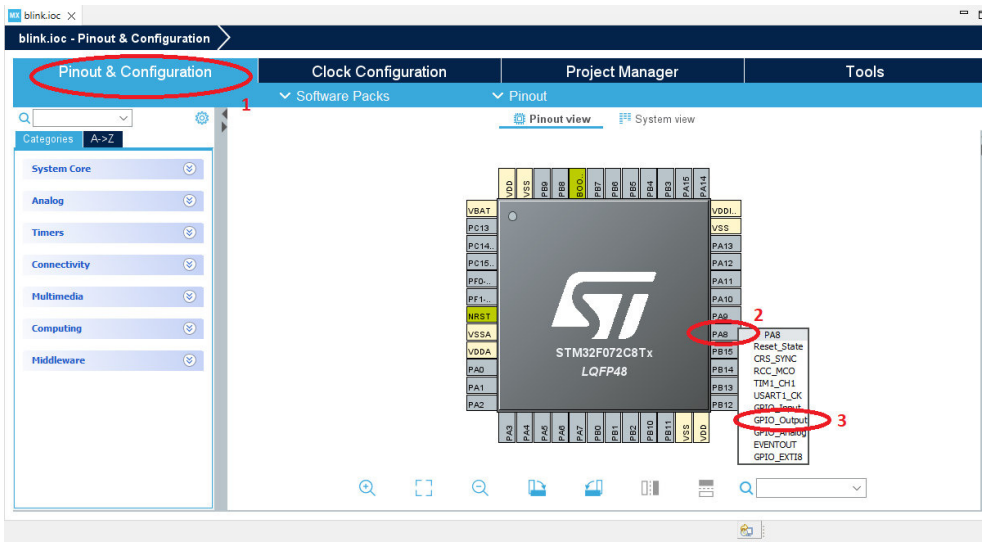


Рис. 2.2. Вибір режиму роботи виводу мікроконтролера в STM32CubeIDE

Решту параметрів у першому проекті залишимо без змін та перейдемо на другу вкладку «Clock Configuration». Тут ми можемо обрати джерело тактування мікроконтролера, задати частоти роботи ядра, шин, периферії. Задамо максимальну частоту тактування 48 МГц (рис. 2.3). Для цього можна обрати або HSI48 RC-генератор з підлаштуванням частоти за USB SOF пакетами, наявний в мікроконтролерах серії STM32F0 з USB-інтерфейсом, або HSI RC-генератор 8 МГц з множенням x6 за допомогою модуля фазового автопідстроювання частоти PLL (Phase-Locked Loop). HSI RC-генератор з частотою 8 МГц наявний в усіх мікроконтролерах серії STM32F0, тому оберемо його, щоб мати налаштування, сумісні з усією серією.

Інші вкладки в першому проекті залишимо без змін.

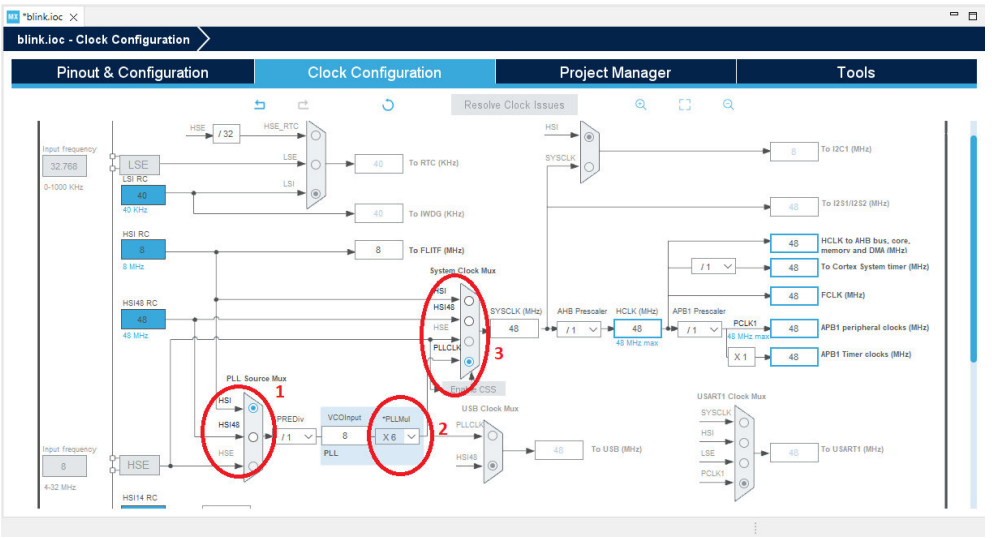


Рис. 2.3. Вибір джерела тактування мікроконтролера

Натискаємо кнопку «Save» або обираємо відповідний пункт у головному меню File->Save, при цьому STM32CubeIDE запропонує згенерувати програмний код – у відповідь погоджуємось. Буде згенеровано проект з програмним кодом у файлі main.c. Розберемо його по частинам.

Весь програмний код у файлах розбитий коментарями /* USER CODE BEGIN */ та /* USER CODE END */ на секції. Під час повторного генерування програмного коду за даними ioc-файла текст програми між коментарями /* USER CODE BEGIN */ та /* USER CODE END */ буде захищено від змін, а решта тексту в файлах проекту буде перезаписано новим згенерованим текстом.

На початку бачимо секцію описом програми й ліцензійними умовами, яку варто модифікувати під свій проект перед його публікацією.

```
/* USER CODE BEGIN Header */
/**
  ******************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
```

```
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ************************************
  */
/* USER CODE END Header */
```

Наступна секція – підключення заголовних файлів. Користувач STM32CubeIDE повинен писати код програми між коментарями /* USER CODE BEGIN */ та /* USER CODE END */, оскільки підключення заголовних файлів, здійснені в іншому місці поза відповідними коментарями, будуть перезаписані (втрачені) при спробі повторно згенерувати код.

```
/* Includes ------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
```

#include "main.h" було додано середовищем STM32CubeIDE, а заголовні файли користувача додаватимуться між рядками /* USER CODE BEGIN Includes */ та /* USER CODE END Includes */.

Наступна секція – секція з псевдонімами типів.

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

/* USER CODE END PTD */
```

Далі стикаємось з секцією для директив #define, які при програмуванні мікроконтролеів часто використовують, щоб задати константу без виділення пам’яті для неї в RAM, а включити її значення в машинний код, розміщений в пам’яті програм. Нехай ми бажаємо замість номера піна 5 порта, до якого підключений світлодіод, використовувати символьний запис LED_PIN. Тоді за потреби змінити номер піна з 5 на 7 можна здійснити заміну лише один раз в секції /* USER CODE BEGIN PD */ #define LED_PIN 5 /* USER CODE END PD */, а не в усіх місцях програми, де він використовується. Того ж ефекту можна досягти, вівши константу const uint8_t led_pin=5, проте для неї буде виділено відповідний байт у оперативній пам’яті й звертання до нього буде тривалішим, ніж до значення, збереженого безпосередньо біля коду інструкції в пам’яті програм. Тому використання #define для проголошення констант при програмуванні мікроконтролерів є поширеною практикою, що дозволяє економити порівняно малий обсяг вбудованої оперативної пам’яті мікроконтролера й на такт-другий пришвидшити виконання програми.

Деякі компілятори на певних платформах автоматично розміщують глобальні константи в пам'яті програм (зокрема arm-none-eabi-gcc в STM32CubeIDE), а на інших платформах в оперативній пам'яті даних (avr-gcc).

Водночас такий підхід не рекомендується при програмуванні ПК без жостких обмежень RAM, характерних для програмування мікроконтролерів – в цій сфері краще надати перевагу проголошенню константи const, а не #define.

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

/* USER CODE END PD */
```

Наступна секція в програмі служить для проголошень макросів.

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

/* USER CODE END PM */
```

Потім одна з найбільш використовуваних секцій – секція проголошення глобальних змінних.

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

/* USER CODE END PV */
```

Далі маємо секцію з прототипами функцій, причому в першій частині будуть розміщені прототипи функцій, створених в результаті генерування програмного коду на основі даних в ioc-файлі, а в другій користувач дописує прототипи власних функцій між рядками /* USER CODE BEGIN PFP */ … /* USER CODE END PFP */.

```
/* Private function prototypes -----------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

Далі передбачена ще одна секція коду /* USER CODE BEGIN 0 */ … /* USER CODE END 0 */ на випадок, якщо користувачу потрібно буде додати перед функцією main() інші рядки коду, що за змістом не підходять у інші секції.

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

/* USER CODE END 0 */
```

Нарешті бачимо реалізацію функції int main(void). Саме ця функція почне виконуватись після запуску програми (є точкою входу в програму) з точки зору користувача. Після запуску мікроконтролера й виконання першої інструкції деякі службові дії будуть виконані перед main(), але їх розгляд виходить за рамки даного практикуму. В більшості випадків користувачу не потрібно виконувати будь-яких додаткових дій, щоб змінити ці дії, але за потреби більш детальніше ознайомитись з ними можна в [2].

```
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

Бачимо першу секцію функції main(), в яку користувач може додавати власний програмний код. Проте варто враховувати, що дана ділянка буде виконана ДО ініціалізації рівня апаратних абстрацій HAL, налаштування тактового генератора й периферії. Після запуску і входу в main() мікроконтролери серії STM32F0 продовжують працювати від внутрішнього RC-генератора частотою 8 МГц до виклику функції SystemClock_Config() нижче. Тому не варто в цій секції розміщати ресурсоємні складні обчислення, якщо надалі мікроконтролер буде переключено на джерело тактування з більшою частотою. Ті самі обчислення, розміщені в програмі після SystemClock_Config(), виконаються швидше.

В цій секції варто розміщати лише ті операції, які обов'язково повинні бути виконані перед ініціалізацією HAL. Також тут варто розмістити проголошення локальних змінних функції main().

```
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

Ще одна секція для коду користувача /* USER CODE BEGIN Init */ … /* USER CODE END Init */ призначена для операцій, які повинні бути виконані одразу після ініціалізації HAL.

```
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
```

```
/* USER CODE END SysInit */
```

Секція /* USER CODE BEGIN SysInit */ … /* USER CODE END SysInit */ призначена для розміщення власного коду ініціалізації та початкової настройки периферійних модулів мікроконтролера. Вона, як і наступні рядки програми, вже виконуватиметься з тактовою частотою, обраною в налаштуваннях у іос-файлі, а не з початковою частотою 8 МГц.

Наступними будуть виконані згенеровані функції ініціалізації периферії, назва яких сформована за шаблоном MX_периферія_Init(). У першому проекті ми використовуємо лише GPIO, тому маємо лише одну функцію ініціалізації MX_GPIO_Init():

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */
```

Засіб генерування коду початкової ініціалізації STM32Cube**MX** історично з'явився раніше STM32CubeIDE (куди був інтегрований), тому в згенерованих функцій наявний перфікс **MX_** .

Секція /* USER CODE BEGIN 2 */ … /* USER CODE END 2 */ використовується для розміщення коду користувача, який виконуватиметься після завершення ініціалізації периферії. Тут можуть бути власні функції ініціалізації зовнішньої периферії, яка підключається до мікроконтролера, а не є його складовою, а отже й не ініціалізується HAL. Наприклад, рідкокристалічні або OLED дисплеї, робота з контролером яких передбачає початкову ініціалізацію, RFID-модулі, цифрові сенсори (BME280, HMC5883L тощо) й інші елементи, що требують ініціалізації перед використанням.

Більшість програм для мікроконтролерів містять основний нескінченний цикл. Хоч фрагмент нижче «while (1) {» і згенерований STM32CubeIDE, користувач може змінити його на свій розсуд, наприклад, задавши умову виходу з циклу чи взагалі видаливши цикл.

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

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

Користувач може додати програмний код, що виконується один раз перед циклом (між рядками /* USER CODE BEGIN WHILE */ та while (1)), і програмний код основного циклу програми, який повинен знаходитись перед /* USER CODE END WHILE */.

Зверніть увагу: пустий рядок між /* USER CODE END WHILE */ та /* USER CODE BEGIN 3 */ не призначений для написання тексту програми користувачем, адже це коментарі-маркери кінця й початку різних секцій. Весь текст програми, розміщений між ними, буде видалено після внесення змін в ioc-файл і повторного генерування програмного коду.

Остання секція /* USER CODE BEGIN 3 */ } /* USER CODE END 3 */ містить лише закриваючу дужку циклу while (1), проте після неї можна додати операції, що мають виконуватись після закінчення головного циклу. Наприклад, після виявлення критичних відхилень у роботі мікропроцесорної системи в основному циклі можна здійснити вихід з циклу з використанням оператора break (if(temperature > temperature_critical) break;), а після циклу передбачити сигналізацію про несправність чи інші дії нештатного завершення роботи системи.

Після функції main() розміщуються реалізації всіх згенерованих функцій. Перша – функція налаштування джерела системного тактового сигналу SystemClock_Config(). Налаштування в HAL передбачає заповнення полів структур RCC_OscInitStruct та RCC_ClkInitStruct значеннями, які відповідають зробленому вибору в графічному режимі.

```
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  /** Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
```

```
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

Якщо налаштування системного тактового генератора закінчилось невдало, викликається функція Error_Handler(), де можна передбачити дії щодо сигналізації даної проблеми користувачеві.

```
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_
CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
    Error_Handler();
  }
}
```

На базі системного тактового сигналу формуються сигнали тактування ядра, пам’яті (шина AHB), шини периферії APB. Якщо налаштування закінчилось невдало, викликається функція Error_Handler(), де можна передбачити дії щодо сигналізації даної проблеми користувачеві.

Наступна функція MX_GPIO_Init() виконує початкове налаштування портів вводу-виводу загального призначення, заповнюючи структуру GPIO_InitStruct й застосовуючи налаштування функцією HAL_GPIO_Init().

```
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

Навідміну від мікроконтролерів архітектури AVR (зокрема ATmega328P в Arduino Uno) тактування кожного порта вводу-виводу повинно бути ввімкнено окремо. Оскільки ми використовуємо контакт PA8, потрібно ввімкнути тактування GPIOA викликом __HAL_RCC_GPIOA_CLK_ENABLE();

Далі відведена секція для функцій користувача, прототипи яких були вказані вище в секції /* USER CODE BEGIN PFP */ … /* USER CODE END PFP */.

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */
```

Зверніть увагу: функції користувача, дописані після функції main(), але перед цією секцією, будуть видалені при внесенні змін в ioc-файл і повторному генеруванні коду.

Однією з останніх є функція Error_Handler(), яка викликається у випадку помилки. Вона містить операції, що виконуватимуться, наприклад, при помилках ініціалізації периферії чи налаштування тактового сигналу.

```
/**
  * @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 */
}
```

За замовчуванням здійснюється вимкнення переривань та вхід у пустий нескінченний цикл, що ззовні для користувача виглядатиме як зависання програми. Можна додати свої операції в цей цикл (наприклад сигналізувати код помилки блиманням світлодіода) або взагалі реорганізувати функцію під більш складну логіку обробки

помилки. Для передачі коду помилки в функцію може використовуватись глобальна змінна.

В коді HAL-бібліотеки є можливість використати assert_param(), що перевіряє допустимість значень аргументів функцій і дозволяє виявити нештатні ситуації (наприклад, передача NULL-вказівника) під час роботи програми. Щоб використати цю можливість, потрібно вказати макрос USE_FULL_ASSERT на початку програми й між рядками /* USER CODE BEGIN 6 */ … /* USER CODE END 6 */ реалізувати тіло функції assert_failed. Саме вона викликатиметься, якщо перевірку не пройдено.

```
#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 */
```

Ознайомившись із згенерованим шаблоном програми, знайдемо функцію main() і додамо включення/виключення світлодіода кожні 1000 мс. Для цього використаємо виклики функцій HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8) та HAL_Delay(1000), які додамо перед коментарем /* USER CODE END WHILE */:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
```

```
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
    HAL_Delay(1000);
   /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Збережемо внесені зміни. Скопілюємо проект, обравши в головному меню Project->Build Project. В результаті компіляції буде створено файл прошивки. Розглянемо детальніше вивід CDT Build Console. Він може бути використаний для формування власної системи збірки прошивок: наприклад, якщо потрібно скомпілювати індивідуальні прошивки для 200 пристроїв з унікальними парами логін-пароль, достатньо буде написати скрипт, що заноситиме логін і пароль в проект і за допомогою make компілюватиме черговий файл прошвики, а не вручну змінювати програму 200 разів.

***** Build of configuration Debug for project blink *****

make -j2 all

Проект збирається за допомогою утиліти make, яка виконує вказані у makefile дії, щоб отримати файл прошивки. Ключ -j використовується, щоб вказати кількість потоків компіляції, які запустить make.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DU SE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/ Include -I../ Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal.d" -MT"Drivers/STM32F0xx_HAL_Driver/Src/stm32f 0xx_hal.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/ Src/stm32f0xx_hal.o"

STM32CubeIDE використовує компілятор GCC для архітектури ARM. Частина «none» у назві вказує, що компілятор створюватиме програми, які виконуватимуться безпосередньо, матимуть монопольний доступ до апаратної частни, а не виконуватимуться у середовищі операційної системи, яка скомпільована й встановлена окремо від програми користувача. Наприклад, програма, скомпільована arm-linux-gnueabi-gcc, повинна виконуватись в операційній системі на базі GNU/Linux і не буде коректно працювати, якщо її просто прошити у flash-пам'ять мікроконтролера без операційної системи.

Першими компілюються HAL-драйвери:

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_cortex.c" -mcpu=cortex-m0 -std=gnu11 -g3 –DDE BUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../ Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_ Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx /Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections –Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_cortex.d" -MT"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_cortex.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_ hal_cortex.o"

Окрім ключів компілятора, які вказують файли з джерельним кодом, каталоги з заголовними файлами, вихідний об'єктний файл, можна відмітити наступні:

-mcpu=cortex-m0 – вказує тип цільового процесора, для якого виконується компіляція програми.

-mfloat-abi=soft – вказує використовувати програмну реалізацію операцій з плаваючою комою замість використання апаратного блоку операцій з плаваючою комою (він відсутній в мікроконтролерах архітектури ARM Cortex-M0).

-mthumb – вказує компілятору використовувати набір більш компактних інструкцій Thumb замість звичайних ARM (A32) ISA. Це дозволяє отримати компактніший код і менший розмір прошивки.

-std=gnu11 – використовувати стандарт мови C C11 (ISO/IEC 9899:2011) з розширеннями GNU.

-O0 – відключити оптимізацію програми. Це спрощує налагодження програми. Значення -O0 встановлюється за замовчуванням при виборі конфігурації побудови «Debug». Інші можливі значення: -O1 - компілятор намагатиметься зменшити розмір коду та час виконання, не виконуючи жодних оптимізацій, які сильно збільшують час компіляції. -O2 - компілятор намагатиметься виконати майже всі підтримувані оптимізації, які не порушують компроміс між швидкістю виконання та розміром виконуваного коду програми. У

порівнянні -O1, цей параметр збільшує як час компіляції, так і продуктивність згенерованого коду. -O3 - включає всі оптимізації -O2, а також дозволяє агресивну оптимізацію, що веде до збільшення розміру виконуваного коду програми, наприклад, розкриття циклів (зникає потреба в інструкції переходу в кінці циклу, але збільшується розмір програми внаслідок повторення коду тіла циклу). -Og - оптимізація для процесу налагодження, прийнятний рівень оптимізації під час розробки (циклів редагування-компіляція-налагодження), що зберігає високу швидкість компіляції та гарні можливості налагодження напротивагу опціям -O1 та -O2. -Os – оптимізація за розміром: дозволяє всі оптимізації рівня -O2, крім тих, які збільшують розмір коду, а також змушує компілятор націлюватись на розмір коду, а не на швидкість виконання, й виконувати додаткові оптимізації, призначені для зменшення розміру коду, застосовувати inline-функції тощо. -Oz - агресивна оптимізація розміру, а не швидкості. Вона може збільшити кількість виконуваних ядром інструкцій, якщо ці інструкції потребують меншої кількості байтів у прошивці. Подібно до -Os застосовуються більшість оптимізацій рівня -O2.

-g3 – задає найвищий рівень (з чотирьох можливих g0...g3) надання налагоджувальної інформації, в результуючий файл включається максимальна кількість налагоджувальної інформації.

-DSTM32F072xB - ключ «-D» передає макрос для препроцесора, в даному випадку інформує про використовуваний в проекті мікроконтролер.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_dma.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/ Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/ Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections –Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_dma.d" -MT"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_dma.o" --specs= nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/ Src/stm32f0xx_ hal_dma.o"

В проект включений файл stm32f0xx_hal_dma.c, хоча поки що ми не використали прямий доступ до пам’яті. Річ в тому, що STM32CubeMX включає у проект всі файли HAL-драйверів, що можуть знадобитись для роботи з вибраною периферією. Подібну ситуацію бачимо й далі.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_exti.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver /Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/ Include -I../ Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections –Wall

-fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal_exti.d" -MT"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_exti.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx _hal_exti.o"

Файл stm32f0xx_hal_exti.c містить функції для роботи з зовнішніми перериваннями EXTI, але поки що в проекті вони не використовуються.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_flash.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver /Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/Include -I../ Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal_flash.d" -MT"Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal_flash.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_flash.o"

Файл stm32f0xx_hal_flash.c та наступний stm32f0xx_hal_flash_ex.c – файли, що містять функції для роботи з Flash-пам’яттю програм. Оскільки мікроконтролери серії STM32F0 не мають окремої вбудованої EEPROM-пам’яті даних, наприклад, для налаштувань готового пристрою (режим роботи, завдання регулятора, коефіцієнти налаштувань ПІД-регулятора), саме функції з цих файлів використовуються для збереження даних у Flash-пам'ять програм (фактично – емуляції EEPROM).

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_flash_ex.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I.. /Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx /Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections –Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_ HAL_Driver/Src/stm32f0xx_hal_flash_ex.d" -MT"Drivers/STM32F0xx _HAL_Driver/Src/stm32f0xx_hal_flash_ex.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_flash_ex.o"

Далі компілюється HAL-драйвер портів вводу-виводу загального призначення GPIO stm32f0xx_hal_gpio.c, функції з якого ми викликали в головному циклі нашої програми.

*arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_gpio.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_nDriver/Inc -I../Drivers/STM32F0xx_HAL_Driver/ Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/Include -I../ Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_hal_gpio.d" -MT"Drivers/STM32F0xx_HAL_Driver /Src/stm32f0xx_*

hal_gpio.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/ STM32F0xx_HAL_Driver/Src/stm32f0xx_hal _gpio.o"

Наступним компілюється файл з HAL-функціями керування живленням stm32f0xx_hal_pwr.c, які надають можливість переводити мікроконтролер у режими зниженого енергоспоживання.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_ pwr.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/ Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx /Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_ hal_pwr.d" -MT"Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_pwr.o" --specs= nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/ stm32f0xx_hal_pwr.o"

Далі компілюється HAL-драйвер stm32f0xx_hal_pwr_ex.c з функціями для роботи з програмованим детектором напруги, який дозволяє відстежувати зміни напруги живлення мікроконтролера, що може бути використано для збереження важливих даних з оперативної в енергонезалежну пам’ять при виявленні тенденції швидкого падіння напруги живлення (вимкнення пристрою).

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_pwr_ex.c" -mcpu=cortex-m0 -std=gnu11 -g3 –DDEB UG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../ Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Drivers/ STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_pwr_ex.d" -MT"Drivers/STM32F0xx_ HAL_Driver/Src/stm32f0xx_hal_pwr_ex.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_ pwr_ex.o"

Наступним компілюється HAL-драйвер stm32f0xx_hal_rcc.c з функціями керування тактуванням мікроконтролера.

arm-none-eabi-gcc "../Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_rcc.c" -mcpu=cortex-m0 -std=gnu11 -g3 –DDEBUG -DUSE _HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc/ Legacy -I../Drivers/CMSIS/Device/ST/STM32F0xx/Include -I../Drivers/CMSIS/ Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD –MP -MF"Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_rcc.d" -MT"Drivers/ STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_rcc.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Drivers/STM32F0xx_HAL_Driver/Src/stm32f0xx_hal_rcc.o"

Після цього формується об’єктний файл, що містить початковий код запуску мікроконтролера, що виконується перед передачею керування в точку входу – функцію main().

arm-none-eabi-gcc -mcpu=cortex-m0 -g3 -DDEBUG -c -x assembler-with-cpp -MMD -MP -MF"Core/Startup/startup_stm32f072 c8tx.d" -MT"Core/Startup/ startup_stm32f072c8tx.o" --specs=nano .specs -mfloat-abi=soft -mthumb -o "Core/Startup/startup_stm32f072 c8tx.o" "../Core/Startup/startup_stm32f072c8tx. s"

Після компілювання HAL-драйверів виконується компіляція файлу з головною функцією програми main.c, в результаті отримуємо об'єктний файл main.o.

arm-none-eabi-gcc "../Core/Src/main.c" -mcpu=cortex-m0 -std =gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/ STM32F0xx/Include -I../Drivers/CMSIS/Include -O0 -ffunction-sectio ns -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Core/Src/ main.d" -MT"Core/Src/main.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Core/Src/main.o"

Далі компілюються інші файли проекту з каталогу Core/Src. stm32f0xx_hal_msp.c – файл з функціями ініціалізації MSP (MCU specific package) і деініціалізації.

arm-none-eabi-gcc "../Core/Src/stm32f0xx_hal_msp.c" -mcpu= cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM 32F072xB -c -I../Core/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver/ Inc/Legacy -I../Drivers/CMSIS/ Device/ST/STM32F0xx/Include -I../Drivers/CMSIS/ Include -O0 –ffunc tion-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Co re/Src/stm32f0xx_hal_msp.d" -MT"Core/Src/stm32f0xx_hal_msp.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Core/Src/stm32 f0xx_hal_msp.o"

Наступний файл stm32f0xx_it.c призначається для обробників переривань і відповідних функцій зворотнього виклику (колбеків, callback-функцій). Навідміну від програмування AVR-мікроконтролерів, де обробники переривань (функції ISR) розміщувались, як правило, в одному файлі з функцією main, у STM32CubeIDE їх прийнятно розміщувати в окремому файлі. Однак при цьому одна глобальна змінна, яка використовується і в обробнику переривань, і в основному файлі main.c, може дублюватись в різних файлах під час компіляції. Тому проголошення змінної в одному файлі має бути з ключовим словом extern, яке вказуватиме компілятору, що пам'ять для цієї змінної вже виділена в іншому місці програми. Інакше ми будемо мати дві незалежні змінні в різних файлах, значення яких жодним чином не пов'язані, оскільки знаходяться і різних комірках пам'яті.

arm-none-eabi-gcc "../Core/Src/stm32f0xx_it.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072 xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../ Drivers/STM32F0xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/ Device/ST/STM32F0xx/Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Co

re/Src/stm32f0xx_it.d" -MT"Core/Src/stm32f0xx_it.o" --specs=nano .specs -mfloat-abi=soft -mthumb -o "Core/Src/stm32f0xx_it.o"

Файл syscalls.c містить реалізацію (в більшості випадків – «заглушки») деяких системних викликів (_fork, _kill, _execve, _exit тощо), які введені для збереження сумісності з кодом мовою С, написаним для систем з операційною системою загального призначення, а не для мікроконтролерів.

arm-none-eabi-gcc "../Core/Src/syscalls.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc/Legacy -I../ Drivers/CMSIS/Device /ST/STM32F0xx/Include -I../Drivers/CMSIS/Include -O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Core/ Src/syscalls.d" -MT"Core/Src/syscalls.o" --specs=nano.specs -mflo at-abi=soft -mthumb -o "Core/Src/syscalls.o"

Файл sysmem.c містить низькорівневі newlib функції, що використовуються для динамічного виділення пам’яті.

arm-none-eabi-gcc "../Core/Src/sysmem.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32F072xB -c -I../Core/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc -I../Drivers/ STM32F0xx_HAL_Driver/Inc/Legacy -I../ Drivers/CMSIS/Device/ST/ STM32F0xx/Include -I../Drivers/CMSIS/Include -O0 -ffunction-sec tions -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Core/Src/ sysmem.d" -MT"Core/Src/sysmem.o" --specs=nano.specs -mfloat-abi=soft -mthumb -o "Core/Src/sysmem.o"

У файлі system_stm32f0xx.c міститься реалізація функції SystemInit(), яка викликається перед функцією main(), а також глобальна змінна SystemCoreClock й функція SystemCoreClockUpdate(), які використовуються при зміні частоти системного тактового сигналу.

arm-none-eabi-gcc "../Core/Src/system_stm32f0xx.c" -mcpu= cortex-m0 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER –DSTM 32F072xB -c -I../Core/Inc -I../Drivers/STM32F0xx_HAL_Driver/Inc -I../Drivers/STM32F0xx_HAL_Driver/ Inc/Legacy -I../Drivers/CMSIS /Device/ST/STM32F0xx/Include -I../Drivers/CMSIS/ Include -O0 –ffun ction-sections -fdata-sections -Wall -fstack-usage -MMD -MP -MF"Core/Src/system_stm32f0xx.d" -MT"Core/Src/system_stm32f0 xx.o" --specs= nano.specs -mfloat-abi=soft -mthumb -o "Core/Src/ system_stm32f0xx.o"

Врешті-решт всі скомпільовані об’єктні файли збираються в ELF-файл, який містить прошивку й дані для налагодження. На його основі можна згенерувати файли прошивки у інших форматах (hex, bin тощо), вони не міститимуть даних для налагодження й саме їх використовують при серійному виготовленні друкованих плат з мікроконтролерами серії STM32F0. З файлів у цих форматах значно важче відновити вихідний код програми, ніж з ELF-файлу з налагоджувальною інформацією. Це актуально у випадку витоку файлу прошивки в треті руки чи спробі недобросовісним підрядником,

який здійснює серійне виробництво плат на замовлення, модифікувати проект і випускати продукт під своїм брендом. Водночас можливість випускати повну копію, маючи файл прошивки, зберігається, тому часто для підрядника-виробника мікропроцесорних плат надається тестова прошивка, яка дозволяє перевірити справність всіх частин, але не містить всіх функцій кінцевого продукту; після отримання протестованих замовлених плат у мікроконтролер завантажується повнофункціональна прошивка й активується захист від зчитування.

arm-none-eabi-gcc -o "blink.elf" @"objects.list" -mcpu=cortex-m0 -T"C:\Users\Admin\STM32CubeIDE\workspace_ 1.10.1\blink\STM32F072C8TX_FLASH.ld" --specs=nosys.specs -Wl, -Map="blink.map" -Wl,--gc-sections -static --specs=nano.specs -mfl oat-abi=soft -mthumb -Wl,--start-group -lc -lm -Wl,--end-group
Finished building target: blink.elf

arm-none-eabi-size blink.elf
arm-none-eabi-objdump -h -S blink.elf > "blink.list"
text data bss dec hex filename
5204 20 1572 6796 1a8c blink.elf
Finished building: default.size.stdout

Finished building: blink.list

13:23:20 Build Finished. 0 errors, 0 warnings. (took 15s.321ms)

Останні рядки виводу результатів компіляції містять обсяг пам'яті, що використовуватиметься прошивкою. У секції Build Analyzer більш наочно можна оцінити використання пам'яті програм і оперативної пам'яті отриманою прошивкою (рис. 2.4).

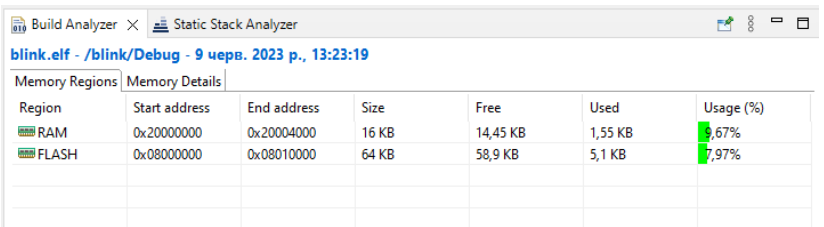


Рис. 2.4. Аналіз використання пам'яті скомпільованою програмою

Отриману прошивку потрібно завантажити в мікроконтролер, для чого можна використати інтерфейс SWD або JTAG. Вони надають можливість як завантаження програми в мікроконтролер, так і налагодження в процесі її виконання. Також прошити мікроконтролер можна через інтерфейси I2C, USB або асинхронний послідовний (модуль USART), запустивши системний завантажувач

мікроконтролера. Проте налагодження програми під час її виконання через ці інтерфейси неможливе.

SWD підтримуються доступними програматорами ST-Link v2, в той час як програматори з підтримкою JTAG мають вищу ціну. Використання зв'язку через USART й I2C потребує на комп'ютері додаткових перетворювачів USB-Serial і USB-I2C. Тому розглянемо завантаження програми в мікроконтролер через інтерфейси SWD і USB.

2.2. Завантаження прошивки в мікроконтролер та налагодження

2.2.1. Прошивка мікроконтролера через SWD

Для завантаження програми в мікроконтролер через інтерфейс SWD потрібен програматор з підтримкою цього інтерфейсу. Далі в книзі буде використовуватись програматор ST-link v2.

Підключення виконується з використанням наступних контактів (програматор->мікроконтролер): 3.3V->VCC, GND->GND, SWCLK->SWCLK (PA14, CLK на платі), SWDIO->SWDIO (PA13, SWD на платі).

Завантаження прошивки через програматор може здійснюватись одразу з середовища STM32CubeIDE. Для цього потрібно обрати в головному меню Run-Run. Перед першим запуском проекту відкриється вікно конфігурації запуску (рис. 2.5). Для першого проекту залишимо всі параметри за замовчуванням та натиснемо кнопку Run.

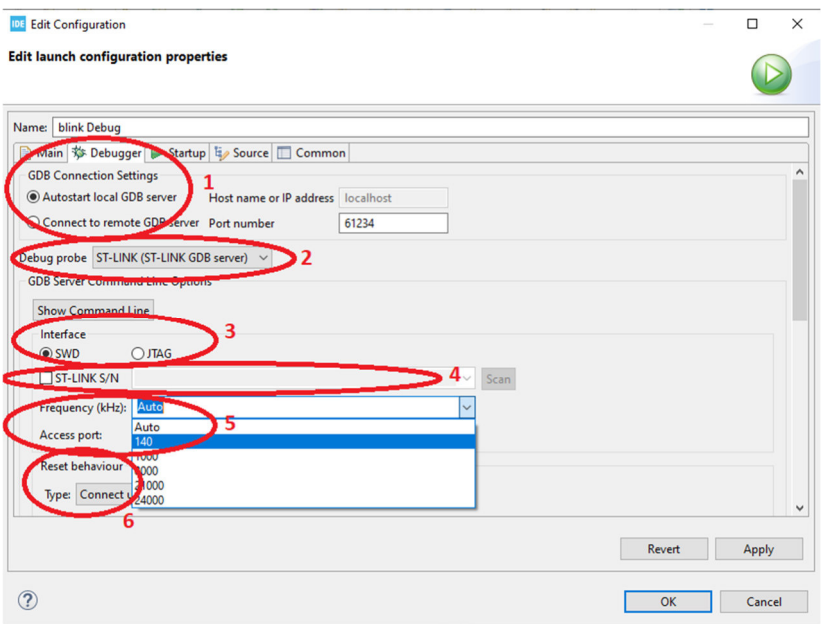


Рис. 2.5. Налаштування конфігурації запуску

Якщо спроба підключитись до мікроконтролера завершилась невдало, перевірте правильність підключення провідників від програматора до плати з мікроконтролером, наявність драйверів програматора в системі, наявність прав відкрити мережевий порт на loopback-інтерфейсів для програми-сервера. Якщо плата підключена до програматора досить довгими провідниками або навколо є джерела сильних електромагнітних полів (напр., працюючий електродвигун чи кабель до потужного споживача електроенергії), використання меншої частоти (наприклад, 140 кГц замість 1000 кГц) може допомогти встановити стабільне підключення (5 на рис. 2.5).

Прошивка відбувається з використанням GDB-сервера, який запускається на локальному комп'ютері (1 на рис. 2.5), проте підтримується також конфігурація з підключенням програматора до віддаленого комп'ютера (в цьому випадку GDB-сервер повинен бути запущений на ньому, а не на комп'ютері зі скомпільованою прошивкою). Для налагодження може використовуватись ST-Link GDB сервер, ST-Link з OpenOCD або Segger J-Link (2 на рис. 2.5). Підтримуються SWD та JTAG як інтерфейси підключення мікроконтролера до програматора (3 на рис. 2.5). Якщо на комп'ютері паралельно виконується програмування одразу кількох мікроконтролерів кількома програматорами, варто прив'язати серійний номер потрібного програматора в конфігурації запуску (4 на рис. 2.5). Частота інтерфейсу програмування за замовчуванням визначається автоматично (5 на рис. 2.5). Максимальна частота залежить від частоти тактування мікроконтролера, довжини провідників, інтенсивності шумів і наводок. Замість повного скидання мікроконтролера при програмуванні можна обрати інший режим скидання (6 на рис. 2.6).

На вкладці Main конфігурації запуску можна додати конфігурацію побудови: Debug або Release. Перша призначена для налагоджденння і під час компіляції проекту не застосовуватиме оптимізації, які погіршують можливості налагодження. Друга призначена для компіляції production-версії прошивки, вона забезпечить компіляцію з оптимізацією розміру прошивки, що іноді дозволяє перейти до використання дешевшого мікроконтролера з меншим обсягом пам'яті програм у серійному мікропроцесорному пристрої/системі.

Виконуючи процедуру запуску програми, STM32CubeIDE виведе у консоль наступні повідомлення:

• Версію GDB-сервера

STMicroelectronics ST-LINK GDB server. Version 7.0.0
Copyright (c) 2022, STMicroelectronics. All rights reserved.

• Опції, з якими запущено GDB-сервер

Starting server with the following options:
Persistent Mode : Disabled
Logging Level : 1
Listen Port Number : 61234
Status Refresh Delay : 15s
Verbose Mode : Disabled
SWD Debug : Enabled
InitWhile : Enabled
Waiting for debugger connection...
Debugger connected

• Версію утиліти програмування й шлях до файлу журналу

STM32CubeProgrammer v2.11.0

Log output file: C:\Temp\STM32CubeProgrammer_a05912.log

• Параметри програматора ST-link

ST-LINK SN : 7
ST-LINK FW : V2J40S7
Board : --
Voltage : 3.26V
SWD freq : 4000 KHz
Connect mode: Under Reset
Reset mode : Hardware reset
Device ID : 0x448
Revision ID : Rev Z
Device name : STM32F07x
Flash size : 64 KBytes
Device type : MCU
Device CPU : Cortex-M0
BL Version : --

Повідомлення в консолі містять напругу живлення на цільовому мікроконтролері, частоту інтерфейсу SWD, який режим підключення та скидання буде використано, ідентифікатор мікроконтролера (відповідає його моделі або лінійці, яка відрізняється розмірами пам’яті).

Далі починається власне завантаження програми в пам'ять мікроконтролера, якому передує стирання відповідних сегментів пам'яті програм.

Memory Programming ...
Opening and parsing file: ST-LINK_GDB_server_a05912.srec
File : ST-LINK_GDB_server_a05912.srec
Size : 5.19 KB
Address : 0x08000000
Erasing memory corresponding to segment 0:
Erasing internal memory sectors [0 2]

Download in Progress:
File download complete
Time elapsed during download operation: 00:00:00.393

Далі здійснюється верифікація: перевіряється, чи співпадає фактичний вміст пам'яті програм із тією програмою, яка тільки-що завантажувалась в мікроконтролер. Успішна верифікація завершує процес завантаження прошивки в мікроконтролер.

Verifying ...
Download verified successfully
Shutting down...
Exit.

2.2.2. Прошивка мікроконтролера через USB

Для завантаження програми в мікроконтролер через інтерфейс USB використовується протокол USB DFU (USB device firmware upgrade). З одного боку на комп'ютері має бути запущена програма STM32CubeProgrammer або DfuSe USB device firmware upgrade, з іншого мікроконтролер повинен бути запущений в режимі оновлення прошивки DFU. Мікроконтролер входить в цей режим у випадку запуску системного завантажувача, а не програми користувача з Flash-пам'яті.

Вибір джерела завантаження здійснюється встановленням логічного нуля чи одиниці на контактах BOOT0 і BOOT1 (див. пункт 2.5. у [3]). Для мікроконтролерів з невеликою кількістю виводів значення біта BOOT1 зберігається в user option byte (біти конфігурації/fuse-біти в термінології інших виробників мікроконтролерів). Для звичайного завантаження з flash-пам'яті програм на контакті BOOT0 у момент виходу мікроконтролера серії STM32F0 з стану Reset (або з режиму Standby) повинна бути напруга низького логічного рівня (лог. 0), значення BOOT1 ігнорується. Якщо на контакт BOOT0 подати логічну одиницю вищезазначений момент і біт BOOT1 рівний нулю, мікроконтролер виконуватиме програму, записану в вбудованій оперативній пам'яті (SRAM). Якщо ж на контакті BOOT0 буде напруга логічної одиниці, а біт BOOT1 встановлений в логічну одиницю, мікроконтролер завантажиться з області системної пам'яті, куди виробником записано завантажувач.

Завантажувач дозволяє отримати нову прошивку з наступних інтерфейсів [4]:

• USART на виводах PA9/PA10, PA14/PA15 або PA2/PA3;
• I2C на виводах PB6/PB7;
• USB DFU інтерфейс.

Саме останній варіант ми будемо використовувати. В новому мікроконтролері встановлено заводське значення біта BOOT1 в

логічну одиницю. Таким чином достатньо подати на контакт BOOT0 логічний нуль, щоб мікроконтролер завантажив користувацьку програму з flash-пам'яті програм, або логічну одиницю, щоб був запущений системний завантажувач замість програми користувача з flash-пам'яті програм.

У платі для виконання лабораторних і практичних робіт для цього використовується група контактів BOOT0: перемичка (джампер), встановлена між центральним контактом і контактом, позначеним «0», відповідає завантаженню мікроконтролера з flash-пам'яті програм, а встановлена між центральним контактом і контактом, позначеним «1», – з системної пам'яті (запуск завантажувача). При використанні інших плат потрібно до контакту BOOT0 підключити через підтягуючий резистор (номінал в діапазоні 3,3-27 кОм) VDD, який підключений до додатного полюса джерела живлення мікроконтролера.

Отже, для прошивки мікроконтролера через USB потрібно подати логічну одиницю на BOOT0 й підключити плату з мікроконтролером до комп'ютера через роз'єм USB. При використанні плати без розпаяного роз'єму USB можна використати конектор USB 2.0 типу A або кабель з ним і підключити їх до мікроконтролера наступним чином: віртуальні землі з'єднуємо (GND->GND, чорний провідник у кабелі USB), контакт мікроконтролера PA12/USB_DP до контакту Data+ (зелений провідник у кабелі USB), контакт мікроконтролера PA11/USB_DM до контакту Data- (білий провідник у кабелі USB), проте «плюс» живлення підключаємо до мікроконтролера НЕ напряму. Напруга на контакті VCC (червоний провідник у кабелі USB) рівна +5 Вольт, що більше допустимої напруги живлення мікроконтролерів STM32. Тому напруга +5 В з USB повинна подаватись на стабілізатор, який понижатиме її до 3,3 В, і лише остання подаватиметься на контакти живлення мікроконтролера.

Перейдемо до програмної частини. Слід завантажити з офіційного сайту https://www.st.com/en/development-tools/stm32cubeprog.html й встановити STM32CubeProgrammer. При підключенні плати з мікроконтролером до комп'ютера повинен з'явитись пристрій з Vendor ID 0483, Product ID DF11, REV 2200. Драйвери для пристрою STM32 BOOTLOADER (рис. 2.6) встановляться разом з STM32CubeProgrammer.

Запустимо STM32CubeProgrammer. Вибираємо спосіб підключення до мікроконтролера USB (рис. 2.7) замість ST-LINK. Якщо на момент запуску програми мікроконтролер не був підключений як STM32 BOOTLOADER, після підключення потрібно оновити список.

Обираємо порт, до якого підключений мікроконтролер і натискаємо «Connect».

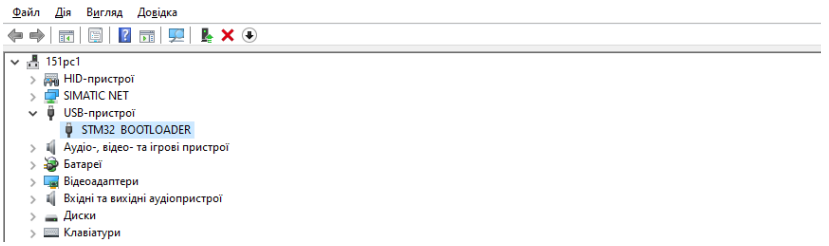


Рис. 2.6. Відображення мікроконтролера у режимі DFU в диспетчері пристроїв

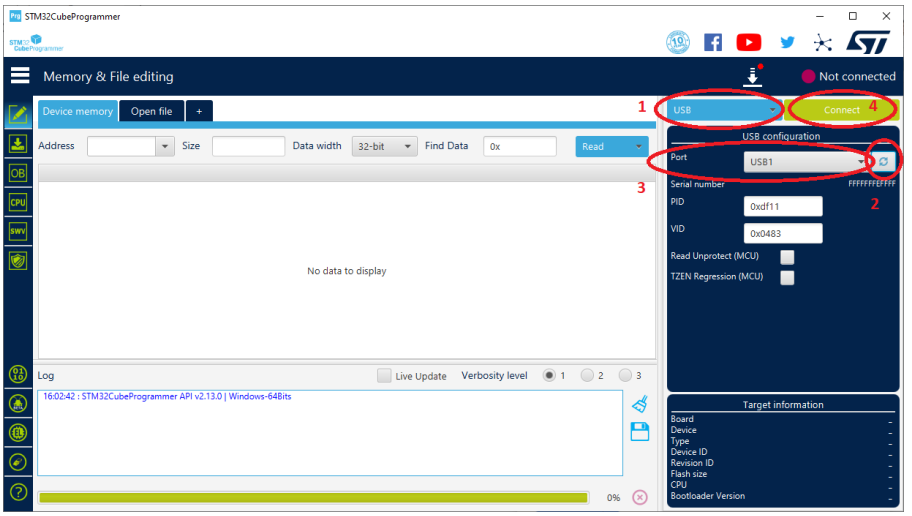


Рис. 2.7. Встановлення зв’язку з мікроконтролером в STM32CubeProgrammer

Коли підключення буде встановлено, STM32CubeProgrammer зчитає поточний вміст пам’яті програм мікроконтролера й відобразить у шістнадцятковій формі. Поряд в стовпчику ASCII відобразяться рядки, отримані якщо розглядати вміст кожного байта пам’яті як ASCII-код символу. Вони дозволяють швидко знайти символьні масиви, збережені в пам’яті. Щоб завантажити нову прошивку, потрібно натиснути по назві вкладки «Open file» (рис. 2.8), обрати відповідний файл прошивки, і в цій вкладці відобразиться вміст файлу.

Далі в списку праворуч обираємо дію «Download» (рис. 2.9), не змінюючи адресу, за якою буде розміщена прошивка у пам’яті мікроконтролера. Адреса 0x08000000 відповідає початку flash-пам’яті програм мікроконтролера (див. розділ 4. Memory mapping у [5]). Нова прошивка запишеться у мікроконтролер.

Залишається відключити конектор USB, переставити перемичку BOOT0 в протилежне положення («0»), щоб при подачі живлення на

мікроконтролер запустилась програма з flash-пам’яті програм, а не системний завантажувач.

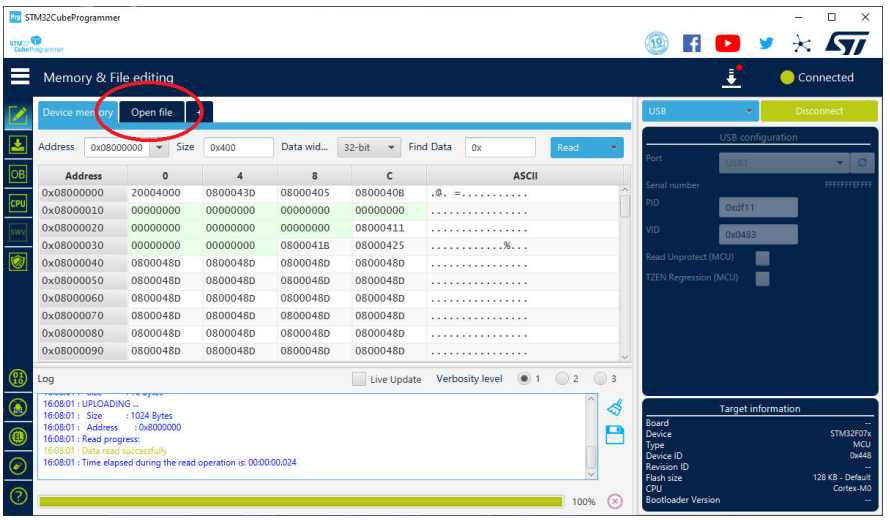


Рис. 2.8. Відкриття файлу з новою прошивкою в STM32CubeProgrammer

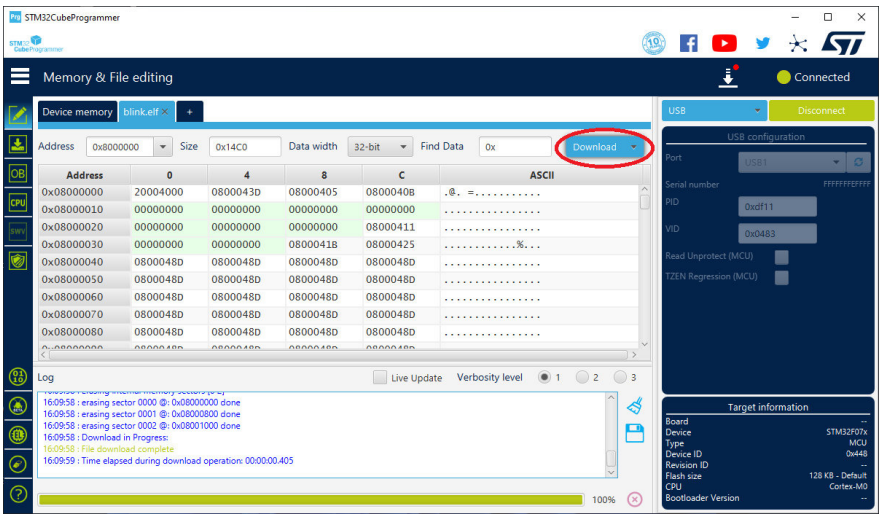


Рис. 2.9. Запуск завантаження нової програми у мікроконтролер

2.2.3. Налагодження програми

Крім звичайного запуску програми й перевірки правильності функціонування мікропроцесорного пристрою (системи) за зовнішніми ознаками (ввімкнення/вимкнення навантаження, робота індикації, адекватність виведених результатів вимірювання аналогових сигналів або опитування цифрових датчиків тощо) при використанні програматора-налагоджувача ST-link можна використовувати функції налагодження: ставити точки зупину програми (брейкпоінти), перевіряти вміст регістрів периферії, ядра, оперативної пам’яті. Це дозволяє бачити стан змінних і регістрів у

будь-який момент без потреби виводити числові значення на індикатор чи послідовний інтерфейс.

Щоб запустити програму в режимі налагодження, потрібно обрати пункт головного меню Run->Debug. Середовище STM32CubeIDE здійснить спробу перемкнутись з перспективи редагування програмного коду до перспективи налагодження, підтверджуємо перехід натисканням «Yes» у відповідному діалоговому вікні. Програма буде завантажена в мікроконтролер, запущена й призупинена на першому рядку функції main() (1 на рис. 2.10).

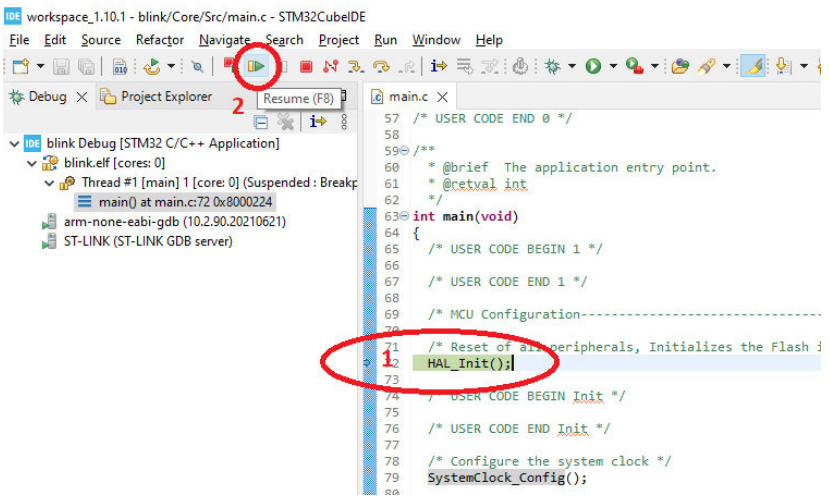


Рис. 2.10. Запущене налагодження у STM32CubeIDE

Відновити роботу призупиненої програми можна, обравши пункт меню Run->Resume або натиснувши відповідну кнопку на панелі інструментів (2 на рис. 2.10).

Перевіримо стан біта 8 вихідного регістра порта GPIO, який задає логічний рівень напруги на виході. Відновивши роботу програми, дочекаємось, коли світлодіод, підключений до PA8, загориться. Це відповідатиме логічній одиниці на виході. Призупинимо програму в цей момент кнопкою «Suspend» або вибором у головному меню Run->Suspend. При цьому відкриється файл (1 на рис. 2.11) з функцією, яка виконувалась у момент надходження команди призупинення, і вказівник поточної інструкції встановиться на рядку, що виконувався в цей момент (2 на рис 2.11). Щоб переглянути вміст регістрів мікроконтролера, переходимо на вкладку SFRs у правій частині вікна (3 на рис. 2.11), знаходимо пристрій STM32F0x2, а в ньому – групу регістрів порта вводу-виводу загального призначення GPIOA (4 на рис. 2.11). Стан виходів визначається вмістом вихідного регістра порта ODR. Залишається знайти потрібний біт ODR8 й пересвідчитись, що його значення дійсно 1.

Аналогічно впевнимось у нульовому значенні біта, коли світлодіод відключений: відновлюємо роботу програми, очікуємо, коли світлодіод погасне, призупиняємо програму й перевіряємо значення того самого біта ODR8, який задає логічний рівень напруги на виході PA8.

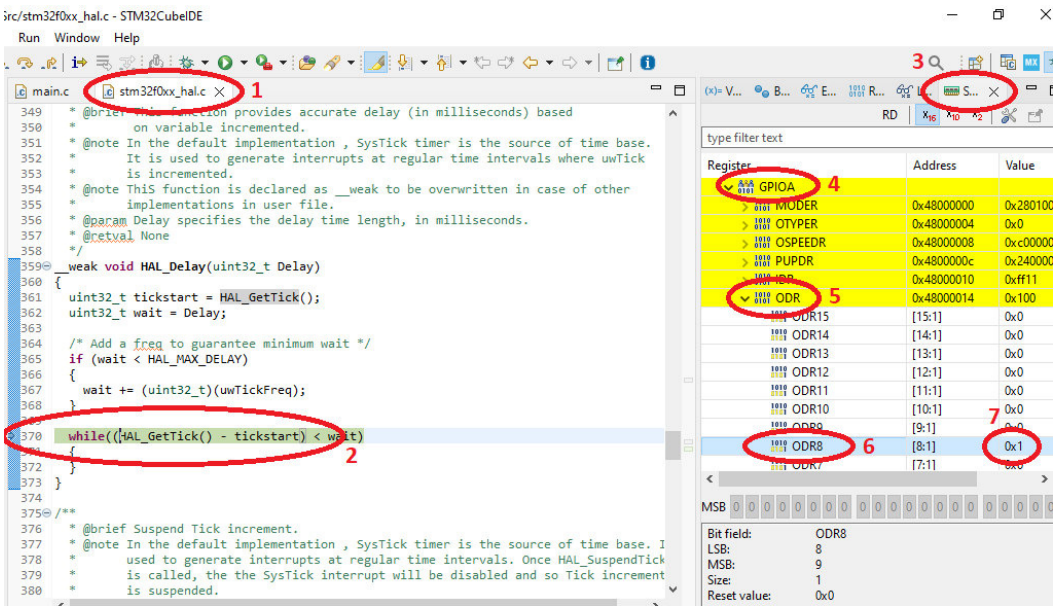


Рис. 2.11. Перегляд значення біта в режимі налагодження

Переглянути значення глобальних змінних (в т.ч. структур) та вказівників, регістрів, які фігурують в програмному коді, можна безпосередньо в тексті програми, навівши курсор на відповідний елемент і затримавши його (рис. 2.12).

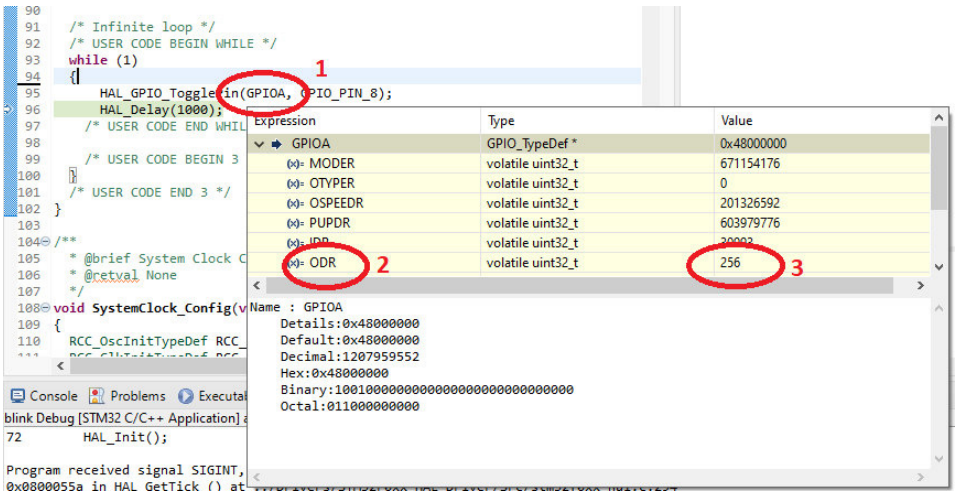


Рис. 2.12. Перегляд значення елемента в програмному коду в режимі налагодження

Щоб зупинити програму перед виконанням конкретного рядка, можна встановити точку зупину (брейкпоінт) на цьому рядку (рис. 2.13). Для цього слід двічі клацнути лівою кнопкою миші по смузі ліворуч нумерації рядків програми. Синя крапка вказуватиме на безумовну точку зупину. Щоб зупинити програму одразу після зміни напруги, що подається на світлодіод, поставимо брейкпоінт одразу після виклику функції HAL_GPIO_TogglePin().

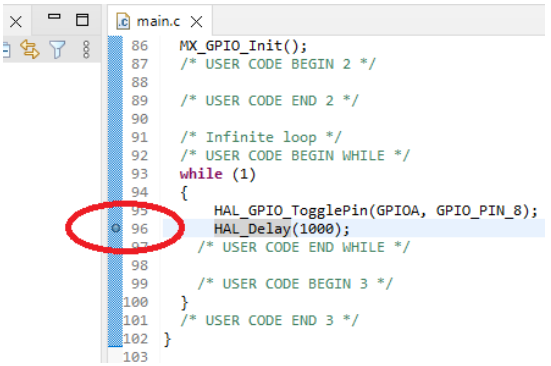


Рис. 2.13. Встановлення брейкпоінта в програмі

Запустимо програму в режимі налагодження, відновимо її роботу після першої зупинки в першому рядку функції main() – виклику функції HAL_Init(). Функція HAL_GPIO_TogglePin() виконається один раз і світлодіод запалиться, після чого виконання програми буде призупинено. Світлодіод продовжуватиме горіти, доки ми не відновимо виконання програми й мікроконтролер знову не виконає функцію HAL_GPIO_TogglePin(), яка погасить світлодіод. Програма знову призупиниться в точці зупину й очікуватиме команди відновлення виконання. Щоб скасувати зупинку програми у вибраному рядку, потрібно зняти брейкпоінт подвійним клацанням по синій крапці.

Крім безумовних брейкпоінтів можна додавати точки зупину, що спрацьовують при виконанні вказаної мови задану кількість разів (наприклад, перевищення певного значення вмістом регістра результату АЦП). Для цього замість подвійного кліку потрібно клацнути правою кнопкою й обрати в контекстному меню «Add Breakpoint».

Також доступні точки спостереження (watchpoint) – точки зупинки, які не прив’язані до конкретних рядків програми, а призупиняють роботу програми при зміні заданого значення (умови) незалежно від того, при виконанні якого рядка програми ця зміна відбулась. Їх додавання доступне на вкладці «Outline» в режимі

редагування коду: потрібно обрати змінну, на зміну стану якої потрібно реагувати, викликати контекстне меню й обрати «Toggle Watchpoint».

Увага! Для використання всіх можливостей налагодження (перегляд значень змінних і регістрів, точки зупину й спостереження) повинна бути обрана конфігруація побудови Debug, а не Release.

3. ЛАБОРАТОРНІ РОБОТИ

3.1. Лабораторна робота 8. Введення-виведення цифрових сигналів мікроконтролером STM32F072C8

Мета роботи: навчитися програмувати мікроконтролери серії STM32F0 для роботи з механічними кнопками та дискретними індикаторними світлодіодами з використанням портів вводу-виводу загального призначення.

Теоретичні відомості

Порти вводу-виводу загального призначення (GPIO) мікроконтролерів родини STM32F0 мають більше режимів роботи [6], між порти мікроконтролерів AVR. Порт (рис. 3.1) може бути налаштовано як:

- “плаваючий” (високоомний) вхід;
- вхід з підтяжкою до напруги живлення (pull-up);
- вхід з підтяжкою до потенціалу [віртуальної] землі (pull-down);
- вихід push-pull;
- вихід з відкритим стоком.

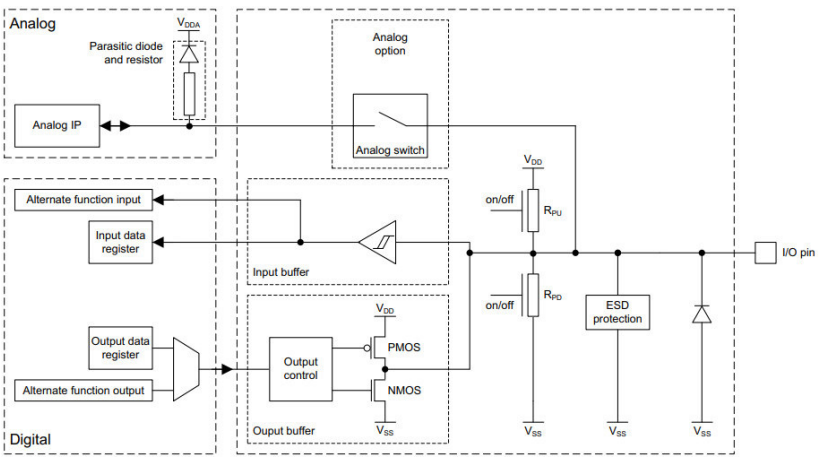


Рис. 3.1. Структура портів вводу-виводу загального призначення типу FT та TT

Режими роботи на вхід з підтяжкою до напруги живлення V_{DD} або з підтяжкою до потенціалу землі V_{SS} задіюють внутрішні підтягуючі резистори опором близько 40 кОм R_{PU} та R_{PD} відповідно.

В режимі push-pull порт працюватиме на вихід аналогічно єдиному вихідному режиму порта AVR-мікроконтролера ATmega328P: відкриватиме n-канальний транзистор NMOS і закриватиме p-канальний PMOS для виведення напруги логічного нуля та

закриватиме n-канальний транзистор NMOS і відкриватиме p-канальний PMOS для виведення напруги логічної одиниці.

Режим виходу з вікдритим стоком задіює лише n-канальний транзистор NMOS, а p-канальний транзистор PMOS залишається завжди закритим. Таким чином вивід мікроконтролера може або підключатись до потенціалу землі V_{SS}, або відключатись і не мати чітко заданого потенціалу. Відповідно, щоб отримати різну напругу при виведенні логічного нуля та одиниці, потрібно задіювати зовнішній або внутрішній підтягуючий резистор до напруги живлення.

Налаштування порта вводу-виводу можна здійснювати безпосереднім записом у кілька регістрів налаштувань GPIO або використати функцію

*HAL_GPIO_Init(GPIOx, *GPIO_InitTypeDef),*

де GPIO_InitTypeDef – вказівник на структуру налаштування порта, GPIOx – порт (GPIOA, GPIOB, …).

В структурі налаштувань порта режим роботи задається значеннями GPIO_MODE_OUTPUT_PP (вихід push-pull), GPIO_MODE_OUTPUT_OD (вихід з відкритим стоком), GPIO_MODE_INPUT (вхід). Конкретизується вхідний режим полем Pull, в яке можна записати значення GPIO_NOPULL («плаваючий» високоомний вхід), GPIO_PULLUP (вхід з підтяжкою до напруги живлення), GPIO_PULLDOWN (вхід з підтяжкою до потенціалу землі).

Стан входу (для прикладу, PC10) можна отримати викликом функції *HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_10)*, яка повертає GPIO_PIN_SET (логічна 1) або GPIO_PIN_RESET (логічний 0).

Змінити стан виходу PC13 можна функціями

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
//скинути в 0
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
//встановити в 1
HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
// перемкнути в протилежний стан
```

Частота портів вводу-виводу загального призначення не задається однозначно частотою тактування обчислювального ядра, як в мікроконтролерах архітектури AVR, а може задаватись низькою (GPIO_SPEED_FREQ_LOW), середньою (GPIO_SPEED_FREQ_MEDIUM), високою (GPIO_SPEED_FREQ_HIGH) за допомогою бітів OSPEEDRx. Конкретні значення частоти вказуються в технічному описі (даташиті, datasheet) відповідного мікроконтролера. В STM32CubeIDE частотою GPIO за замовчуванням є низька, що при напрузі живлення 3,3 В, згідно таблиці 55 «I/O AC characteristics» в технічному описі [5],

відповідає 2 МГц. Якщо напруга живлення менше 2 В, відповідні частоти будуть менші.

Порти вводу-виводу загального призначення відрізняються за своїми характеристиками. Звичайні порти вводу-виводу, розраховані на напругу 3,3 В, позначаються в технічному описі TC і не витримують вхідної напруги 5 В. Порти, позначені TTa, мають пряме підключення до АЦП й теж розраховані на напругу 3,3 В. Порти, що можуть працювати з вхідною напругою 5 В, позначаються FT (5 V-tolerant I/O).

Всі порти вводу-виводу, за вийнятком SWDIO та SWCLK, під час та після скидання мікроконтролера налаштовуються як високоомні входи, напруга на яких не буде стабільною в непідключеному стані. Електромагнітні поля, що нас оточують, змінюватимуть напругу на високоомному вході, що призводитиме до хаотичного перемикання вхідного буфера порту то в стан логічної одиниці при підвищенні напруги до рівня V_{IH} (див. таблицю 53 «I/O static characteristics» в технічному описі [5]), то в стан логічного нуля при падінні напруги до V_{IL}. Непідключений високоомний вхід споживатиме більший струм, ніж вхід, на якому встановлена напруга одного логічного рівня. Щоб уникнути зайвих перемикань стану входів, зменшити стрибки струму споживання мікроконтролера та його абсолютне значення, на непідключених входах варто активувати внутрішній підтягуючий резистор до потенціалу землі чи живлення або підключити цей вхід до «плюса» чи «мінуса» живлення мікроконтролера. Це забезпечить стабільний сигнал на вході.

Підключення кнопок та світлодіодів до мікроконтролера STM32F072C8T6 здійснюється за тими ж схемами (рис. 3.2), що й AVR-мікроконтролерів (див. лабораторну роботу 1).

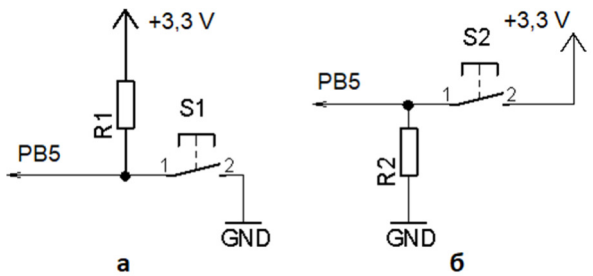


Рис. 3.2. Схеми підключення кнопок до мікроконтролера: а) з підтяжкою до напруги живлення; б) з підтяжкою до потенціалу землі

У схемі з підтяжкою до напруги живлення натиснутій кнопці на вході мікроконтролера відповідає напруга логічного нуля, відпущеній – логічної одиниці. У схемі з підтяжкою до землі натиснутій кнопці на

вході мікроконтролера відповідає напруга логічної одиниці, відпущеній – логічного нуля.

План роботи

1. Реалізувати періодичне ввімкнення-вимкнення навантаження, підключеного до контакту PA8.

2. Зчитати стан кнопки на контакті PB5 та просигналізувати його за допомогою світлодіода, підключеного до контакту PA8.

3. Реалізувати зміну періоду блимання світлодіода двома кнопками, підключеними до контактів PC14 та PC15.

Порядок виконання роботи

1. Запустити STM32CubeIDE, створити новий проект мовою C й налаштувати в ioc-файлі на вихід контакт PA8, до якого підключити світлодіод: анод світлодіода підключити до виходу мікроконтролера, катод – через струмообмежувальний резистор опором 470 Ом…2,2 кОм до GND. Використовуючи функції *void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)* та *void HAL_Delay (uint32_t Delay)*, реалізувати блимання світлодіода з частотою 1 Гц. Аргумент Delay – кількість мілісекунд, на які треба затримати виконання програми. Задамо тривалість ввімкнення 700 мс та тривалість вимкнення та 300 мс, що в сумі складе період 1000 мс й дасть частоту 1 Гц. Замінимо вміст нескінченого циклу в функції main() з попередньої програми на два виклики HAL_GPIO_WritePin(), розділені двома викликами функції затримки HAL_Delay(). Отримаємо наступний програмний код головної функції:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
```

```
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_Delay(700);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
    HAL_Delay(300);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Використовується той самий вихід PA8, що й в першому проекті, тому тіло функції MX_GPIO_Init() співпадатиме з наведеним раніше.

Скомпілювати проект, завантажити програму в мікроконтролер, в режимі налагодження зупинити програму й пересвідчитись, що запаленому стану світлодіода відповідає 1 у регістрі ODR, а погашеному – 0.

2. Підключити кнопку до контакту PB5 (контакт INT на групі контактів GY-521) та налаштувати підтяжку до напруги живлення (рис. 3.2, а). Для цього в ioc-файлі двічі клацнути по контакту PB5, обрати режим GPIO_Input (рис. 3.3).

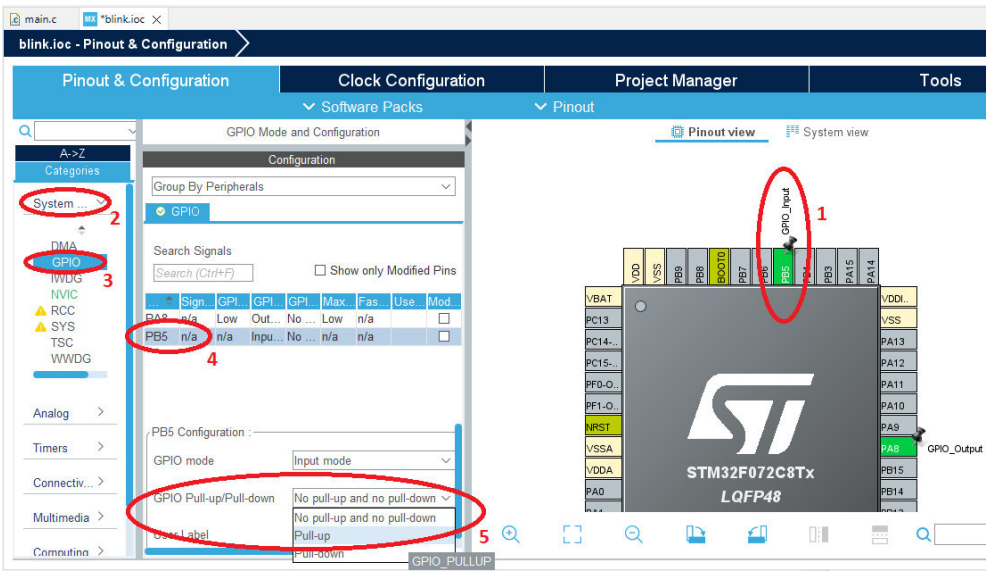


Рис. 3.3. Налаштування контакту PB5 як входу з підтяжкою до напруги живлення

Далі ліворуч обрати розділ «System Core», в ньому – GPIO, в отриманій таблиці виділити рядок з контактом PB5, обрати в списку властивості «GPIO Pull-up/Pull-down» варіант «Pull-up». Зберегти зміни й згенерувати оновлений програмний код. В результаті функція MX_GPIO_Init() має набути вигляду

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

Для зчитування логічного сигналу на вході використаємо функцію *GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin).* Далі модифікуємо основний цикл програми так, щоб при надходженні логічного нуля на вхід PB5 запалювався світлодіод, підключений до контакту PA8. Для цього додамо умову, що порівняє значення, повернуте функцією HAL_GPIO_ReadPin, з GPIO_PIN_RESET (натиснутій кнопці відповідає низький рівень напруги), і, якщо умова істинна, виводить високий рівень напруги (логічну одиницю) на вихід PA8 викликом HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET). Якщо умова не виконується, викликаємо цю ж саму функцію, але з останнім аргументом GPIO_PIN_RESET. У результаті функція main() набуде вигляду:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_5) == GPIO_PIN_RESET)
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
    else
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Скомпілювати проект, прошити мікроконтролер і перевірити реакцію мікроконтролера на натиснення кнопки.

3. Змінити схему й активувати підтяжку до потенціалу спільного провідника (віртуальної землі) згідно рис. 3.2, б. Відповідно в ioc-файлі в списку властивості «GPIO Pull-up/Pull-down» слід обрати варіант «Pull-down». Тепер натиснутій кнопці відповідатиме високий рівень напруги на вході, а відпущеній – низький. Змінюємо значення, з яким порівнюється значення, повернуте функцією HAL_GPIO_ReadPin(), на протилежне (GPIO_PIN_SET). В результаті умовний оператор матиме вигляд

```
if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_5) == GPIO_PIN_SET)
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
else
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
```

Скомпілювати проект, прошити мікроконтролер і перевірити реакцію мікроконтролера на натиснення кнопки, підключеної за іншою схемою.

4. Реалізувати збільшення частоти блимання світлодіода кнопкою Btn1 (вхід PC14), а зменшення – Btn3 (PC15). Кнопки Btn1 та Btn3 одним контактом з'єднані з GND/Vss, відповідно повинні використовуватись з підтяжкою до напруги живлення. Знову змінимо ioc-файл, вказавши в списку властивості «GPIO Pull-up/Pull-down» варіант «Pull-up». Функція MX_GPIO_Init() має набути вигляду

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC13 PC14 PC15 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

Далі передбачимо змінну, що зберігатиме значення затримки в мілісекундах. Розмістимо її проголошення у відповідній секції:

```
/* Private variables ---------------------------------------*/
/* USER CODE BEGIN PV */
int delayMs=500;
/* USER CODE END PV */
```

Початкову затримку задамо 500 мс. Додамо умови, яка проаналізують стан обох кнопок. По натисканню однієї кнопки будемо збільшувати затримку на 100 мс, іншої – зменшувати. Передбачимо обмеження зміни затримок у допустимих межах. Функція main() набуде вигляду:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14)==GPIO_PIN_RESET)
            if(delayMs>100) delayMs-=100; //не зменшувати тривалість включення/виключення, якщо в результаті отримаємо замале значення
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15)==GPIO_PIN_RESET)
            if (delayMs<5000) delayMs+=100; //не збільшувати тривалість вище максимально допустимої, яку тут прийнято 5100 мс
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
    HAL_Delay(delayMs);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Перевірку двох умов за допомогою двох умовних операторів можна здійснити в одному, об’єднавши умови логічним «І». Тоді тіло головного циклу спроститься:

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

```
    {
      if((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == GPIO_PIN_RESET) &&
(delayMs > 100)) delayMs-=100;
      if((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET) &&
(delayMs < 5000)) delayMs+=100;
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
      HAL_Delay(delayMs);
      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
```

Скомпілювати проект, завантажити програму в мікроконтролер, в режимі налагодження перевірити реакцію мікроконтролера на натискання кнопок, переконатись в зміні значення змінної, що задає величину затримки.

5. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Які засоби введення/виведення інформації користувача використовуються в мікропроцесорній техніці?
2. Які схеми використовуються при підключенні кнопок до мікропроцесорних пристроїв?
3. Які схеми використовуються при підключенні окремих світлодіодів до мікропроцесорних пристроїв?
4. Які види «підтяжок» доступні в мікроконтролерах родини STM32F0?
5. Як зчитати стан входу за допомогою бібліотеки HAL для мікроконтролерів STM32?
6. Як змінити стан виходу на протилежний за допомогою бібліотеки HAL для мікроконтролерів STM32?
7. Як змінити стан виходу на заданий за допомогою бібліотеки HAL для мікроконтролерів STM32?
8. В якому вхідному режимі має працювати вхід, щоб зчитувати вихідний сигнал датчика з виходом типу «сухий контакт»?

Завдання для самостійної роботи

1. Реалізувати позиційне керування рівнем рідини у резервуарі з двома дискретними датчиками рівня з виходом типу «відкритий колектор». При надходженні на контакт PC14 логічної одиниці від дискретного давача нижнього рівня (рівень впав нижче нижньої межі) вмикати насос подачі рідини у резервуар сигналом з контакту PA8 (активний рівень сигналу – високий), а при надходженні на контакт

PC15 логічного нуля від дискретного давача нижнього рівня (рівень досяг верхньої межі) – вимикати насос.

2. Модифікувати програму з п. 3 порядку виконання лабораторної роботи № 8 так, щоб період опитування кнопок не залежав від поточної затримки. Використати функцію HAL_GetTick(), попередньо ознайомившись з способом її використання в функції void HAL_Delay(uint32_t Delay) з файлу stm32f0xx2_hal.c. Щоб швидко відкрити файл з реалізацією функції, клацніть по виклику функції HAL_Delay() в головному циклі, викличте контекстне меню й оберіть «Open Declaration».

3. Інфрачервоний давач руху HC-SR501 підключений до цифрового входу PB5. Написати програму для мікроконтролера, яка на виході PA8 генерує імпульси ввімкнення навантаження тривалістю 5 секунд з періодом 15 секунд. При виявленні руху вищезазначеним давачем блокувати ввімкнення навантаження на 60 секунд.

4. Скласти програму для контролера світлофора, в якого зміна фаз ініціюється натисканням кнопки пішоходом. Затримка після натискання кнопки перед зміною фаз 15 секунд. Тривалість ввімкнення зеленого сигналу для пішоходів – 20 секунд. Тривалість жовтого кольору – 3 секунди. Масив світлодіодів червоного кольору для автомобілів керується з контакту мікроконтролера PB15, жовтого – PB14, зеленого – PB13, червоного кольору для пішоходів – PB12, зеленого для пішоходів – PA6. Активний рівень для ввікмнення – високий (логічна одиниця). На платі для лабораторних робіт це відповідно контакти MOSI, MISO, SCK, SS в восьмиконтактному роз’ємі RC522, контакт «+» в двоконтактному роз’ємі I/O.

5. У програмі з попереднього завдання передбачити зміну затримки перед зміною фаз світлофора після натискання кнопки пішоходом залежно від часу, що пройшов з моменту попереднього вимкнення червоного кольору для автомобілів. Передбачити наступну залежність: якщо пішохід натиснув кнопку в момент ввімкнення зеленого кольору для автомобілів або раніше (коли автомобілям горів червоний з жовтим, а пішоходу – вже червоний) – затримка 45 секунд, якщо пішохід натиснув кнопку, коли пройшло 60 секунд від моменту включення зеленого кольору для автомобілів, – затримка 10 секунд. Між крайніми значеннями 0 секунд та 60 секунд затримка змінюється за лінійним законом. Якщо пройшло більше 60 секунд – затримка 10 секунд.

3.2. Лабораторна робота 9. Портування програми регулювання для STM32F072C8

Мета роботи: навчитися портувати програми для мікроконтролерів; навчитися використовувати АЦП мікроконтролерів серії STM32F0 в режимі неперервного перетворення.

Теоретичні відомості

Під час життєвого циклу мікропроцесорного пристрою може виникнути задача заміни мікроконтролера в його основі на інший внаслідок зміни цінової політики виробника мікроконтролера, появи дешевших замінників, зміни вимог до функціональності пристрою, виявлення апаратних недоліків. Це вимагає портування старої програми на новий пристрій. Портування в межах однієї родини мікроконтролерів може обмежитись переналаштування портів вводу-виводу, пов'язаних зі апаратними змінами у друкованій платі. Якщо в новому мікроконтролері використовуються відмінні модулі (наприклад, 10-бітний АЦП з програмованим підсиленням замість 10-бітного АЦП без нього), потрібно вносити зміни й у ділянки програми, пов'язані з цією периферією.

Портування на інші родини або навіть інші архітектури потребує більш значних змін. Може змінитись розмір доступної пам'яті, глибина стеку, доступні режими сну, частоти. Інша архітектура може вимагати використання інших компіляторів, які передбачають відмінну розмірність типів даних, інший синтаксис роботи з бітами тощо.

STM32F072C8 містить 12-бітний АЦП з можливістю сканування групи каналів, неперервної роботи й запуску по події-тригеру, генеруванням переривання про готовність результату, а також передачею результату через DMA. Бібліотека HAL містить функції для роботи з АЦП, імена яких починаються на HAL_ADC_, зокрема:

Ініціалізація модуля АЦП

*HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef * hadc),*

де hadc — вказівник на хендл модуля АЦП.

Повертає HAL_OK при успішній ініціалізації; HAL_ERROR при помилці.

Налаштування каналу АЦП

*HAL_StatusTypeDef HAL_ADC_ConfigChannel(ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig),*

де sConfig — вказівник на структуру конфігурації каналу

```
typedef struct
{
 uint32_t Channel;    // номер каналу
```

uint32_t Rank; // послідовність сканування каналу, в STM32F0 не викор.

uint32_t SamplingTime; // час вибірки

} ADC_ChannelConfTypeDef;

Запуск модуля АЦП

*HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef * hadc),*

де hadc — вказівник на хендл модуля АЦП.

Повертає HAL_OK при успішному запуску; HAL_ERROR при помилці.

Очікування закінчення перетворення

*HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef * hadc, uint32_t Timeout),*

де Timeout — таймаут в мілісекундах.

Запуск модуля АЦП з генеруванням переривання

*HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef * hadc)*

Запуск модуля АЦП з передачею результату за допомогою DMA

*HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length),*

де pData – вказівник на буфер даних, Length – кількість даних.

Зчитування результату перетворення (ADC_DR)

*uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef * hadc)*

Зчитує черговий канал АЦП (в STM32F0 в порядку зростання номера каналу)

План роботи

1. Налаштувати 12-бітний АЦП мікроконтролера STM32F072C8T6.

2. Портувати програму двопозиційного регулювання з індикацією темпратури, написану для AVR-мікроконтролера ATmega328P, на мікроконтролер STM32F072C8T6 архітектури ARM Cortex-M0.

Порядок виконання роботи

1. Запустити середовище розробки STM32CubeIDE, створити проект мовою С, в ioc-файлі налаштувати АЦП на виконання неперервного перетворення напруги з ADC5 з максимальним часом вибірки 239,5 циклів (рис. 3.4). Щоб в регістрі результату АЦП завжди було найсвіжіше значення, задамо параметру «Overrun behaviour» значення «Overrun data overwritten». Це призведе до перезапису попереднього результату аналогово-цифрового перетворення, якщо він не зчитаний, новим результатом вимірювання напруги. На вкладці Clock Configuration задамо системну тактову частоту 48 МГц.

Збережемо зміни й згенеруємо програмний код. Функція ініціалізації АЦП повинна набути вигляду:

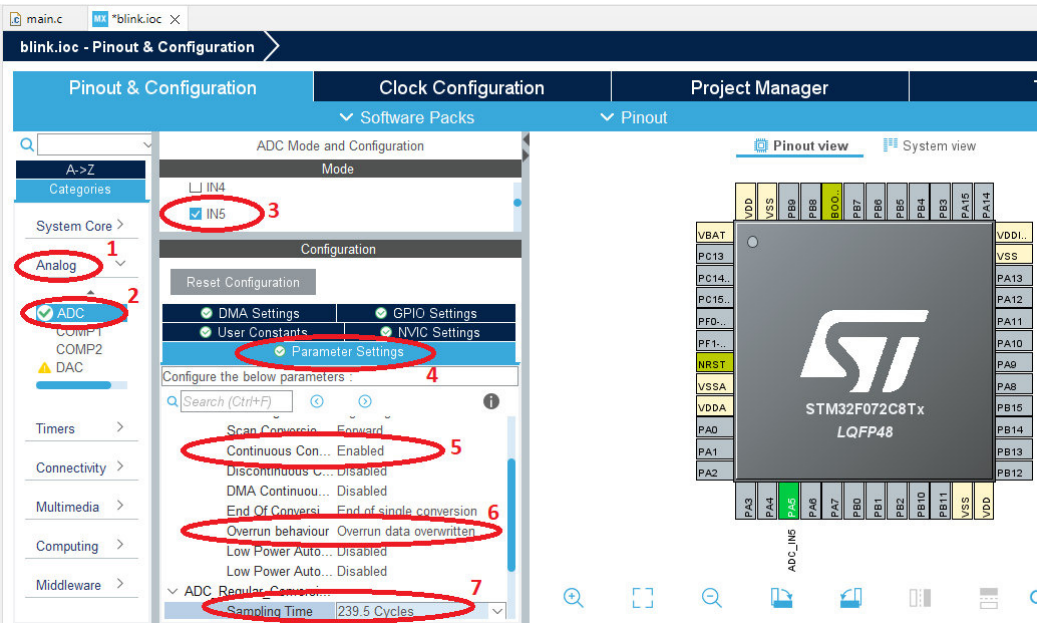


Рис. 3.4. Налаштування модуля АЦП в STM32CubeIDE

```
static void MX_ADC_Init(void)
{
  /* USER CODE BEGIN ADC_Init 0 */

  /* USER CODE END ADC_Init 0 */
  ADC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = ENABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
```

```
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */

  /* USER CODE END ADC_Init 2 */
}
```

В функцію main() додамо проголошення змінної int adc_res=0 для збереження зчитаного результату аналогово-цифрового перетворення, запустимо АЦП в режимі неперервного перетворення перед нескінченим циклом викликом функції HAL_ADC_Start(&hadc), де hadc – хендл модуля АЦП. У тілі циклу потрібно буде почекати готовності результату аналогово-цифрового перетворення викликом фунції HAL_ADC_PollForConversion(&hadc, 100), тут 100 – таймаут очікування результату. Врешті, залишається зчитати результат у створену раніше змінну: adc_res = HAL_ADC_GetValue(&hadc).

Функція main() матиме вигляд:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
```

```
  MX_ADC_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start(&hadc);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_ADC_PollForConversion(&hadc, 100);
    adc_res = HAL_ADC_GetValue(&hadc);
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Скомпілювати програму й завантажити її в мікроконтролер. У режимі налагодження перевірити значення змінної adc_res: поки до аналогового входу ADC_IN5 нічого не підключено, результати аналогово-цифрового перетворення хаотично змінюватимуться, оскільки фактично на вході АЦП матимемо шум.

2. Перевіривши роботу АЦП, налаштуємо таймер. У програмі для AVR використовувався 16-розрядний таймер 1, а точна частота його спрацювання задавалась регістром OCR1A в режимі порівняння. У мікроконтролерах STM32 маємо можливість задати число, до якого рахуватиме таймер, в звичайному режимі.

У мікроконтролері STM32F072C8T6 є 9 таймерів (див. таблицю 7 «Timer feature comparison» в технічному описі [5]). Оскільки нам не потрібно генерувати сигнал з таймера на вихід мікроконтролера, можна не використовувати таймери загального призначення або таймер розширеного керування, а вибрати базовий таймер TIM6 або TIM7. Це 16-розрядні таймери з 16-розрядним передподільником. Оберемо таймер TIM6.

Розрахуємо коефіцієнт передподільника та «стелю» лічби таймера, при досягненні якої він переповнюватиметься (задається в регістрі TIM6_ARR). Задамось частотою оновлення всього динамічного світлодіодного дисплею 100 Гц. Оскільки дисплей трьохрозрядний, частота перемикання розрядів повинна бути 300 Гц. Розрахуємо, скільки тактів системного тактового сигналу 48 МГц відповідає періоду перемикання розрядів індикатора. 48000000 / 300 = 160000. Отримане число займає більше, ніж 16 двійкових розрядів (2^{16}=65536), а отже не може бути записане в регістр автозавантаження таймера TIM6_ARR. Поділимо на 4 тактовий сигнал на вході таймера за допомогою передподільника. Тоді в регістр TIM6_ARR потрібно

записати число 160000 / 4 = 40000. Згідно п. 21.4.7 технічного опису [5] частота на виході передподільника f = f_{CK_PSC} / (PSC[15:0] + 1). Тому в регістр TIM6_PSC слід записати значення 4 – 1 = 3, щоб отримати ділення частоти на 4. Задамо відповідні параметри в ioc-файлі (рис. 3.5).

Переривання від таймера TIM6 можна дозволити на вкладці «NVIC Settings», відмітивши чекбокс «TIM6 global and DAC channel underrun error interrupts». Тепер кожну 1/300 секунди викликатиметься функція *void TIM6_DAC_IRQHandler(void)* з файлу stm32f0xx_it.c (файл для розміщення обробників переривань). У цій функції бачимо єдиний рядок – виклик функції HAL_TIM_IRQHandler(&htim6), оскільки переривання від цифро-аналогового перетворювача (DAC) не дозволені. Функція HAL_TIM_IRQHandler() перевіряє, що спричинило переривання (подія захоплення чи порівняння на каналі 1, 2..., подія оновлення/переповнення, інші події) й викликає відповідну функцію-колбек.

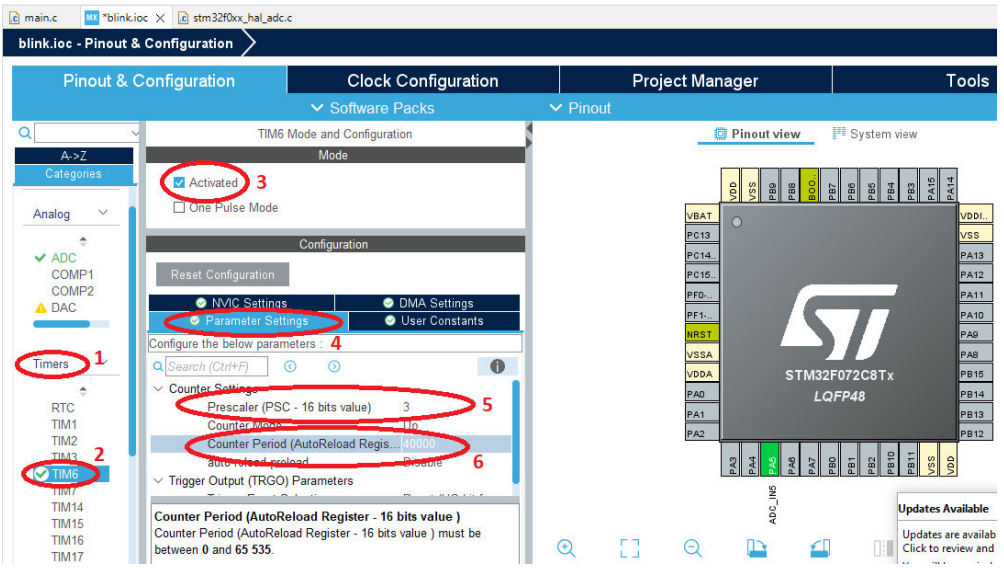


Рис. 3.5. Налаштування таймера TIM6

У випадку переповнення генерується подія оновлення й викличеться колбек *void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)*. Саме цю функцію повинен реалізувати користувач у своєму проекті, щоб виконати певні дії при переповненні таймера.

Збергіаємо ioc-файл і генеруємо оновлений програмний код. Тепер у проекті повинна з'явитись функція ініціалізації таймера TIM6:

static void MX_TIM6_Init(void)

{

```
  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 3;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 40000;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */
}
```

Таймер функцією MX_TIM6_Init() буде налаштовано, але не запущено. Запустити таймер потрібно викликом функції HAL_TIM_Base_Start_IT(&htim6), який слід розмістити до початку головного циклу. Функція запустить базовий таймер, на який вказує хендл htim6 з генеруванням переривань.

2. Налаштувати на вихід контакт PA8 – вихід регулятора, до якого підключити світлодіод. В ioc-файлі налаштувати на вихід (GPIO_Output) контакти світлодіодного динамічного індикатора, підключеного до мікроконтролера, вказані в табл. 3.1.

Таблиця 3.1

Контакт індикатора	Контакт мікроконтролера
DIG1	PB4
DIG2	PB3
DIG3	PB0

продовження табл. 3.1

A	PF0
B	PA15
C	PA9
D	PB9
E	PB8
F	PF1
G	PB2
DP	PA10

В результаті зроблених налаштувань отримуємо наступні використані входи-виходи мікроконтролера (рис. 3.6):

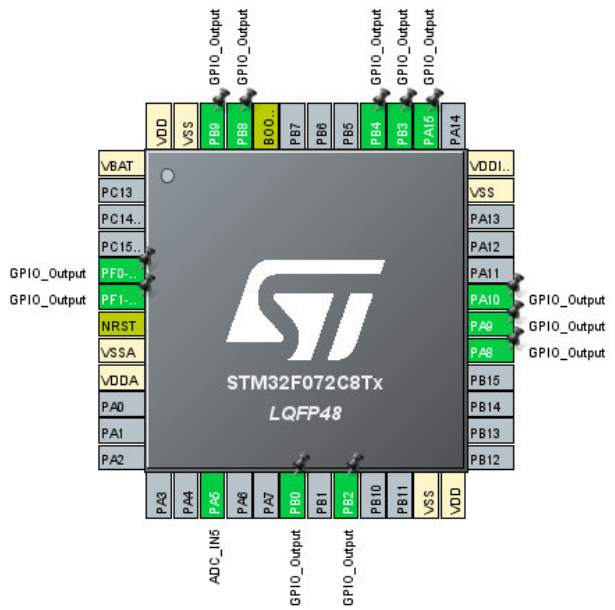


Рис. 3.6. Контакти мікроконтролера, задіяні в лабораторній роботі

Збережемо зміни в ioc-файлі, згенеруємо оновлений код програми й отримаємо наступну функцію ініціалізації портів вводу-виводу загального призначення (GPIO):

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0|GPIO_PIN_1, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_0|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_8|GPIO_PIN_9,
GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|
GPIO_PIN_15,  GPIO_PIN_RESET);
  /*Configure GPIO pins : PF0 PF1 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

  /*Configure GPIO pins : PB0 PB2 PB3 PB4
                         PB8 PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                        |GPIO_PIN_8|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pins : PA8 PA9 PA10 PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

3. Портувати програму з лабораторної роботи 4, яка реалізує двопозиційний регулятор температури з індикацією, на ARM-мікроконтролер. До аналогового входу мікроконтролера подаватиметься напруга з подільника напруги, утвореного термістором 10 кОм, B3950 та постійним резистором 7,5 кОм.

З лабораторної роботи 4 маємо наступну програму:

```
#define F_CPU 16000000UL //тактова частота AVR-мікроконтролера для
формування затримок функцією _delay_ms()
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/io.h>
```

```
#include <avr/pgmspace.h>
// Значення температури, що повертається, якщо сума результатів АЦП більше першого значення таблиці
#define TEMPERATURE_UNDER -250
// Значення температури, що повертається, якщо сума результатів АЦП менше останнього значення таблиці
#define TEMPERATURE_OVER 1250
// Значення температури, що відповідає першому значенню таблиці
#define TEMPERATURE_TABLE_START -250
// Крок таблиці
#define TEMPERATURE_TABLE_STEP 20
// Тип кожного елемента в таблиці, якщо сума виходить у межах 16 біт – uint16_t, інакше – uint32_t
typedef uint16_t temperature_table_entry_type;
//Тип індексу таблиці. Якщо в таблиці більше 256 елементів, то uint16_t, інакше - uint8_t
typedef uint8_t temperature_table_index_type;
// Метод доступу до елемента таблиці повинен відповідати temperature_table_entry_type
#define TEMPERATURE_TABLE_READ(i) pgm_read_word(&termo_table[i])
/* Таблиця сумарного значення АЦП залежно від температури. Від більшого значення до меншого
   для побудови таблиці використані такі параметри:
   R1(T1): 10кОм (25°C)
   B25/100: 3950
   Ra: 7.5кОм
   Напруги U0/Uref: 5В/5В
*/
const temperature_table_entry_type termo_table[] PROGMEM = {
  31149, 30942, 30713, 30463, 30189, 29890, 29566, 29215,
  28838, 28432, 27998, 27537, 27047, 26529, 25985, 25415,
  24821, 24204, 23566, 22910, 22237, 21551, 20854, 20148,
  19438, 18725, 18012, 17303, 16600, 15905, 15221, 14549,
  13893, 13252, 12629, 12025, 11441, 10878, 10335, 9814,
  9314, 8836, 8379, 7942, 7527, 7132, 6756, 6399,
  6061, 5740, 5436, 5149, 4877, 4619, 4376, 4147,
  3930, 3725, 3532, 3350, 3177, 3015, 2862, 2717,
  2580, 2451, 2330, 2215, 2106, 2003, 1907, 1815,
  1729, 1647, 1569, 1496
};

// Функція обчислює значення температури у десятих частках градусів Цельсія
// Залежно від сумарного значення АЦП.
 int16_t calc_temperature(temperature_table_entry_type adcsum) {
```

```
        temperature_table_index_type l = 0;
        temperature_table_index_type r = (sizeof(termo_table) /
sizeof(termo_table[0])) - 1;
        temperature_table_entry_type thigh = TEMPERATURE_TABLE_READ(r);

        // Перевірка виходу межі та граничних значень
        if (adcsum <= thigh) {
          #ifdef TEMPERATURE_UNDER
            if (adcsum < thigh)
              return TEMPERATURE_UNDER;
          #endif
          return TEMPERATURE_TABLE_STEP * r + TEMPERATURE_TABLE_START;
        }
        temperature_table_entry_type tlow = TEMPERATURE_TABLE_READ(0);
        if (adcsum >= tlow) {
          #ifdef TEMPERATURE_OVER
            if (adcsum > tlow)
              return TEMPERATURE_OVER;
          #endif
          return TEMPERATURE_TABLE_START;
        }

        // Двійковий пошук по таблиці
        while ((r - l) > 1) {
          temperature_table_index_type m = (l + r) >> 1;
          temperature_table_entry_type mid = TEMPERATURE_TABLE_READ(m);
          if (adcsum > mid) {
            r = m;
          } else {
            l = m;
          }
        }
        temperature_table_entry_type vl = TEMPERATURE_TABLE_READ(l);
        if (adcsum >= vl) {
          return l * TEMPERATURE_TABLE_STEP + TEMPERATURE_TABLE_START;
        }
        temperature_table_entry_type vr = TEMPERATURE_TABLE_READ(r);
        temperature_table_entry_type vd = vl - vr;
        int16_t res = TEMPERATURE_TABLE_START + r *
TEMPERATURE_TABLE_STEP;
        if (vd) {
          // Лінійна інтерполяція
          res -= ((TEMPERATURE_TABLE_STEP * (int32_t)(adcsum - vr) + (vd >> 1)) /
vd);
        }
```

```
  return res;
}

//volatile забороняє компілятору оптимізовувати звертання до змінної, значення якої могло змінитись в перериванні
volatile uint16_t n=0;
volatile int8_t digit=1, disp[3]={5,3,3};

//Функція для формування зображення цифри, вказаної аргументом d
void showDigit(uint8_t d) {
switch(d) {case 0:
    PORTD|=1<<PD5; //сегмент A
    PORTD|=1<<PD7; //сегмент B
    PORTC|=1<<PC2; //сегмент C
    PORTC|=1<<PC4; //сегмент D
    PORTC|=1<<PC5; //сегмент E
    PORTD|=1<<PD6; //сегмент F
    PORTD&=~(1<<PD4); //сегмент G
    PORTC&=~(1<<PC3); //сегмент DP
    break;
case 1:
    PORTD&=~(1<<PD5);
    PORTD|=1<<PD7;
    PORTC|=1<<PC2;
    PORTC&=~(1<<PC4);
    PORTC&=~(1<<PC5);
    PORTD&=~(1<<PD6);
    PORTD&=~(1<<PD4);
    PORTC&=~(1<<PC3);
    break;
case 2:
    PORTD|=1<<PD5;
    PORTD|=1<<PD7;
    PORTC&=~(1<<PC2);
    PORTC|=1<<PC4;
    PORTC|=1<<PC5;
    PORTD&=~(1<<PD6);
    PORTD|=1<<PD4;
    PORTC&=~(1<<PC3);
    break;
case 3:
    PORTD|=1<<PD5;
    PORTD|=1<<PD7;
    PORTC|=1<<PC2;
    PORTC|=1<<PC4;
```

```
        PORTC&=~(1<<PC5);
        PORTD&=~(1<<PD6);
        PORTD|=1<<PD4;
        PORTC&=~(1<<PC3);
        break;
    case 4:
        PORTD&=~(1<<PD5);
        PORTD|=1<<PD7;
        PORTC|=1<<PC2;
        PORTC&=~(1<<PC4);
        PORTC&=~(1<<PC5);
        PORTD|=1<<PD6;
        PORTD|=1<<PD4;
        PORTC&=~(1<<PC3);
        break;
    case 5:
        PORTD|=1<<PD5;
        PORTD&=~(1<<PD7);
        PORTC|=1<<PC2;
        PORTC|=1<<PC4;
        PORTC&=~(1<<PC5);
        PORTD|=1<<PD6;
        PORTD|=1<<PD4;
        PORTC&=~(1<<PC3);
        break;
    case 6:
        PORTD|=1<<PD5;
        PORTD&=~(1<<PD7);
        PORTC|=1<<PC2;
        PORTC|=1<<PC4;
        PORTC|=1<<PC5;
        PORTD|=1<<PD6;
        PORTD|=(1<<PD4);
        PORTC&=~(1<<PC3);
        break;
    case 7:
        PORTD|=1<<PD5;
        PORTD|=1<<PD7;
        PORTC|=1<<PC2;
        PORTC&=~(1<<PC4);
        PORTC&=~(1<<PC5);
        PORTD&=~(1<<PD6);
        PORTD&=~(1<<PD4);
        PORTC&=~(1<<PC3);
        break;
```

```
        case 8:
            PORTD|=1<<PD5;
            PORTD|=1<<PD7;
            PORTC|=1<<PC2;
            PORTC|=1<<PC4;
            PORTC|=1<<PC5;
            PORTD|=1<<PD6;
            PORTD|=(1<<PD4);
            PORTC&=~(1<<PC3);
            break;
        case 9:
            PORTD|=1<<PD5;
            PORTD|=1<<PD7;
            PORTC|=1<<PC2;
            PORTC|=1<<PC4;
            PORTC&=~(1<<PC5);
            PORTD|=1<<PD6;
            PORTD|=1<<PD4;
            PORTC&=~(1<<PC3);
            break;
            }
        }
        //переривання по співпадінню вмісту таймера з OCR1A для перемикання
розрядів динамічного світлодіодного індикатора
        ISR(TIMER1_COMPA_vect) {
        switch(digit) {  //засвічуємо заряд, вказаний в digit
        case 1:
            PORTB&=~(1<<PB1); //засвічування першого розряду
            PORTD|=1<<PD2; //погашення другого розряду
            PORTB|=1<<PB0; //погашення третього розряду
            break;
        case 2:
            PORTB|=1<<PB1; //погашення першого розряду
            PORTD&=~(1<<PD2); //засвічування другого розряду
            PORTB|=1<<PB0; //погашення третього розряду
            break;
        case 3:
            PORTB|=1<<PB1; //погашення першого розряду
            PORTD|=1<<PD2; //погашення другого розряду
            PORTB&=~(1<<PB0); //засвічування третього розряду
            break;
            }
        showDigit(disp[digit-1]); //виводимо цифру з масиву disp у відповідний
розряд, враховуючи, що нумерація розрядів 1…3, а елементів у масиві 0…2
```

```
    digit++; //збільшуємо номер розряду для засвічування  при наступному спрацюванні таймера
    //якщо номер розряду більший за кількість розрядів – починаємо спочатку, з першого розряду
    if(digit>3) digit=1;
    }

    int main() {
        int sp=280; // 28.0 °C задана темпераиура
        int gist=30; // 3.0 °C (+/-1,5 °C) гістерезис
        DDRB=0b00100011; //вихід на світлодіод/реле на 13-му контакті (PB5)
        DDRC=0b00111100;
        DDRD=0b11110100;
    //налаштування таймера 1
        TCCR1A=0;
        TCNT1=0;
        TCCR1C=0;
        TIMSK1 = 0b10; //дозволити переривання при події порівняння
        OCR1AH=0x7a; //рахувати до
        OCR1AL=0x11; //31249
        TCCR1B = 0b01001; // активувати режим 4
        sei(); //дозволити переривання
        //налаштування АЦП для вимірювання напруги з каналу ADC0
        ADMUX=0b01000000;
        ADCSRA=0b10000111;
        ADCSRB=0b00000000;

        while(1) {
          int res=0;
          //виконуємо 32 послідовних опитування АЦП для фільтрації шумів усередненням
          for(uint8_t i=0; i<32; i++)
          {
            ADCSRA|=1<<ADSC;
            while(!(ADCSRA&(1<<ADSC)));
            res+=ADCL|(ADCH<<8);
            }
          //розраховуємо температуру на основі суми результатів аналогово-цифрового перетворення каналу ADC0
          int16_t temperature=calc_temperature(res);
          //виконуємо двопозиційне регулювання з зоною гістерезису, вихід – PB5
          if(temperature>sp+gist/2) PORTB&=~(1<<PB5); //якщо температура вища заданої – вимикаємо нагрів
```

```
            if(temperature<sp-gist/2) PORTB|=1<<PB5; //якщо температура
менша заданої – вмикаємо нагрів
            //розбиваємо значення температури на розряди й виводимо на
динамічний світлодіодний індикатор
            disp[0]=temperature/100; //лівий розряд («сотні»)
            disp[1]=(temperature-disp[0]*100)/10; //центральний розряд
(«десятки»)
            disp[2]=temperature%10; //правий розряд («одиниці»)
            _delay_ms(200);
            }
        return 0;
    }
```

Очікувано на платі з іншим мікроконтролером зміняться назви й номери портів вводу-виводу, замість прямого запису в вихідні регістри портів використовуватимуться HAL-функції. Також зміняться частини програми, що стосуються налаштування таймера, АЦП, запуску і зчитування результату АЦП. Водночас функція розрахунку температури, позиційне регулювання, робиття на розряди для вивденення на дисплей можуть бути перенесені практично без змін.

Після виконання попередніх пунктів маємо налаштовану архітектурно-специфічну частину програми: вже виконано налаштування (функція MX_TIM6_Init()) і запуск (функція HAL_TIM_Base_Start_IT()) таймера TIM6, налаштовано, запущено й отримано результат АЦП (функції MX_ADC_Init(), HAL_ADC_Start(), HAL_ADC_PollForConversion(), HAL_ADC_GetValue()), налаштовано порти вводу-виводу загального призначення (функція MX_GPIO_Init()). Залишається перенести в STM32CubeIDE функцію розрахунку температури calc_temperature(), функцію засвічування цифри showDigit(), помістити в обробник переривання по переповненню таймера TIM6 тіло функції ISR(TIMER1_COMPA_vect), перенести тіло основного циклу програми з реалізацією опитування АЦП, розрахунком температури, двопозиційним регулюванням й розбиттям температури на розряди для індикації.

Почнемо з функції calc_temperature(). Оскільки в STM32F072C8T6 один адресний простір для команд і даних, глобальні константи зберігаються компілятором в flash-пам’яті програм і звертання до них здійснюється так само, як до даних в оперативній пам’яті (різниця лише в адресі), відпадає потреба в функції зчитування даних в flash-пам’яті програм. Перенесемо в секцію

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

/* USER CODE END PTD */
```

всі рядки з typedef з програми для AVR і в секцію

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

 /* USER CODE END PD */
```

всі рядки з #define, модифікувавши останній: замість доступу до значення в flash-пам'яті програм функцією pgm_read_word()

```
#define TEMPERATURE_TABLE_READ(i) pgm_read_word (&termo_table[i])
```

використаємо звичайний доступ до елемента масиву в пам'яті

```
#define TEMPERATURE_TABLE_READ(i) (termo_table[i])
```

В секцію

```
/* Private variables ---------------------------------------------*/
ADC_HandleTypeDef hadc;
TIM_HandleTypeDef htim6;
/* USER CODE BEGIN PV */
int adc_res=0;

/* USER CODE END PV */
```

перенесемо проголошення масиву termo_table[], видаливши атрибут PROGMEM.

В секцію після функції main() /* USER CODE BEGIN 4 */… /* USER CODE END 4 */ перенесемо реалізацію функції calc_temperature(), а її прототип розмістимо в секції

```
/* Private function prototypes -----------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

Перенесемо функцію showDigit(). Встановлення логічної одиниці на виході AVR-мікроконтролера (запис на зразок PORTD|=1<<PD5) замінимо викликом HAL-функції з останнім аргументом GPIO_PIN_SET (з врахуванням даних таблиці 3.1): HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET). Відповідно подача низької напруги на контакт індикатора здійснюватиметься функцією HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET). Для схеми з мікроконтролером STM32F072C8T6 функція showDigit() запишеться:

```
void showDigit(uint8_t d) {
switch(d) {
   case 0:
      HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
```

```
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
    break;
case 1:
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
    break;
case 2:
     HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
    break;
case 3:
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
    break;
case 4:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
```

```
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        break;
    case 6:
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        break;
    case 7:
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        break;
    case 8:
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        break;
    case 9:
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
```

```
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
            break;
        }
    }
```

Наступним перенесемо обробник переривання таймера. У файл з обробниками переривань stm32f0xx_it.c в секцію /* USER CODE BEGIN 1 */ … /* USER CODE END 1 */.

Додамо реалізацію функції *void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)* з перемиканням розрядів світлодіодного динамічного семисегментного індикатора. Аналогічно випадку з функцією showDigit() відповідно до табл. 3.1 додамо HAL-функції цифрового виводу й отримаємо

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  switch(digit) {  //засвічуємо заряд, вказаний в digit
  case 1:
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_RESET);
//засвічування першого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_SET);
//погашення другого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_0,  GPIO_PIN_SET);
//погашення третього розряду
        break;
  case 2:
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_SET);
//засвічування першого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_RESET);
//погашення другого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_0,  GPIO_PIN_SET);
//погашення третього розряду
        break;
  case 3:
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_SET);
//засвічування першого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_SET);
//погашення другого розряду
HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_0,  GPIO_PIN_RESET);
//погашення третього розряду
        break;
        }
```

showDigit(disp[digit-1]); //виводимо цифру з масиву disp у відповідний розряд, враховуючи, що нумерація розрядів 1…3, а елементів у масиві 0…2

//запалюємо десяткову крапку на другому розряді

if(digit==2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);

else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);

digit++; //збільшуємо номер розряду для засвічування при наступному спрацюванні таймера

//якщо номер розряду більший за кількість розрядів – починаємо спочатку, з першого розряду

if(digit>3) digit=1;

}

Оскільки функція обробки переривання реалізована у іншому файлі, для звертання до змінних і функцій з файлу main.c потрібно використати ключове слово extern, яке вказуватиме, що це функція чи змінна з іншого файлу, відповідно пам'ять під змінну вже виділена при компіляції іншого файлу (main.c). В файлі stm32f0xx_it.c додамо в секцію глобальних змінних:

/ Private variables ---*/*

/ USER CODE BEGIN PV */*

uint8_t digit=1;

extern uint8_t disp[3];

/ USER CODE END PV */*

Оскільки змінна digit використовується лише в цьому файлі, проголошення й виділення пам'яті здійснимо тут, а масив disp [3] заповнюється значеннями в main.c, тому в stm32f0xx_it.c залишимо вказівку компілятору, що цей масив із зовнішнього файлу (extern), а в файлі main.c у секцію /* USER CODE BEGIN PV */ … /* USER CODE END PV */ додамо власне проголошення масиву:

uint8_t disp[3]={1,2,3};

Перейдемо до переносу тіла функції main(). Для полегшення налагодження програми замість проголошення змінної temperature як локальної в функції main(), проголосимо її глобально, адже значення глобальних змінних можна легко перевірити в процесі налагодження. Додамо в секцію /* USER CODE BEGIN PV */ … /* USER CODE END PV */ рядок

int16_t temperature=0;

При реалізації кількаразового послідовного опитування для усереднення результатів і зменшення впливу шумів розрядність АЦП і кількість повторних опитувань визначають значення в таблиці відповідності «сума АЦП – температура», а відповідно й в масиві termo_table[ ]. При 10-бітному АЦП і 32 (2^5) ітераціях циклу опитування сума результатів АЦП 15-бітна. Якщо стоїть мета не перераховувати заново таблицю, то 15-бітну суму 12-бітних результатів АЦП можна

отримати, якщо підсумовувати не 32 послідовні опитування, а 8 (2^3). Якщо кількість опитувань треба зберегти на рівні 32, то доведеться збільшити розрядність змінної для збереження суми, адже $2^{12}2^5=2^{17}$. В такому випадку змінимо тип змінної temperature на int32_t й замінимо значення в масиві termo_table[ ] на наступні:

```
const temperature_table_entry_type termo_table[] = {
  124597, 123767, 122853, 121851, 120755, 119561, 118264, 116862,
  115350, 113728, 111993, 110146, 108187, 106117, 103941, 101661,
  99284, 96816, 94265, 91639, 88949, 86203, 83414, 80593,
  77750, 74898, 72048, 69212, 66399, 63619, 60883, 58197,
  55570, 53008, 50518, 48102, 45766, 43511, 41341, 39256,
  37256, 35343, 33514, 31770, 30108, 28526, 27023, 25596,
  24242, 22959, 21744, 20594, 19506, 18478, 17506, 16588,
  15720, 14901, 14128, 13399, 12710, 12060, 11446, 10867,
  10321, 9805, 9318, 8858, 8424, 8014, 7626, 7260,
  6914, 6587, 6277, 5985
};
```

Розрахунок таблиці наведено в додатку Ґ.

З врахуванням вищевказаного, початок тіла основного циклу запишеться

```
adc_res = 0;
  for(uint8_t i=0; i<8; i++) {
        HAL_ADC_PollForConversion(&hadc, 100);
        adc_res += HAL_ADC_GetValue(&hadc);
  }
temperature = calc_temperature(adc_res);
```

У змінній adc_res маємо суму за 8 або 32 опитування АЦП, яку передаємо як аргумент функції розрахунку температури. Функцію розрахунку температури в умовах більшої кількості пам’яті та вищої швидкодії мікроконтролерів STM32 порівняно з AVR можна реалізувати не лише на основі табличного пошуку, а й аналітичного розрахунку (див. AppNote 1753 [7]). Це вимагатиме арифметичних операцій з плаваючою комою й може зайняти довший час.

Переносимо програмну реалізацію регулювання, змінивши прямий запис у вихідний регістр порта AVR-мікроконтролера викликом HAL-функції HAL_GPIO_WritePin().

```
if (temperature<sp-gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
if (temperature>sp+gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
```

Програмний код розбиття на розряди й запису в масив для індикації переносимо без змін:

```
disp[0]=temperature/100;
disp[1]=temperature/10-disp[0]*10;
```

```
disp[2]=temperature%10;
```

Залишається замість виклику функції програмної затримки _delay_ms(200) додати HAL_Delay(200).

В результаті файл main.c міститиме секції з наступним програмним кодом:

```
/* Private typedef -----------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef uint16_t temperature_table_entry_type;
typedef uint8_t temperature_table_index_type;
/* USER CODE END PTD */

/* Private define ------------------------------------------------*/
/* USER CODE BEGIN PD */
#define TEMPERATURE_UNDER -250
#define TEMPERATURE_OVER 1250
#define TEMPERATURE_TABLE_START -250
#define TEMPERATURE_TABLE_STEP 20
#define TEMPERATURE_TABLE_READ(i) (termo_table[i])
/* USER CODE END PD */

/* Private variables ---------------------------------------------*/
ADC_HandleTypeDef hadc;
TIM_HandleTypeDef htim6;
/* USER CODE BEGIN PV */
int temperature=0;
uint8_t disp[3]={1,2,3};
int adc_res=0;
int sp=280; // 28.0 °C задана темпераиура
int gist=30; // 3.0 °C (+/-1,5 °C) гістерезис
const temperature_table_entry_type termo_table[] = {
   31149, 30942, 30713, 30463, 30189, 29890, 29566, 29215, 28838, 28432,
27998, 27537, 27047, 26529, 25985, 25415, 24821, 24204, 23566, 22910, 22237,
21551, 20854, 20148, 19438, 18725, 18012, 17303, 16600, 15905, 15221, 14549,
13893, 13252, 12629, 12025, 11441, 10878, 10335, 9814, 9314, 8836, 8379, 7942,
7527, 7132, 6756, 6399, 6061, 5740, 5436, 5149, 4877, 4619, 4376, 4147, 3930, 3725,
3532, 3350, 3177, 3015, 2862, 2717, 2580, 2451, 2330, 2215, 2106, 2003, 1907, 1815,
1729, 1647, 1569, 1496 };
/* USER CODE END PV */

/* Private function prototypes -----------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
```

```
int16_t calc_temperature(temperature_table_entry_type adcsum);
void showDigit(uint8_t d);
/* USER CODE END PFP */

int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start(&hadc);
  HAL_TIM_Base_Start_IT(&htim6);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    adc_res=0;
    for(uint8_t i=0; i<8; i++) {
            HAL_ADC_PollForConversion(&hadc, 100);
            adc_res += HAL_ADC_GetValue(&hadc);
    }
    temperature=calc_temperature(adc_res);
    if (temperature<sp-gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_SET);
    if (temperature>sp+gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_RESET);
    disp[0]=temperature/100;
    disp[1]=temperature/10-disp[0]*10;
    disp[2]=temperature%10;
```

```
    HAL_Delay(200);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

У stm32f0xx_it.c файлі повинні бути наступні секції:

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
uint8_t digit=1;
extern uint8_t disp[3];
/* USER CODE END PV */

/* Private function prototypes -----------------------------------*/
/* USER CODE BEGIN PFP */
extern void showDigit(uint8_t d);
/* USER CODE END PFP */

/* USER CODE BEGIN 1 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  switch(digit) {  //засвічуємо заряд, вказаний в digit
  case 1:
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_RESET);
//засвічування першого розряду
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_SET);
//погашення другого розряду
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_0,  GPIO_PIN_SET);
//погашення третього розряду
        break;
  case 2:
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_SET);
//засвічування першого розряду
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_RESET);
//погашення другого розряду
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_0,  GPIO_PIN_SET);
//погашення третього розряду
        break;
  case 3:
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_4,  GPIO_PIN_SET);
//засвічування першого розряду
                HAL_GPIO_WritePin(GPIOB,  GPIO_PIN_3,  GPIO_PIN_SET);
//погашення другого розряду
```

```
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
//погашення третього розряду
            break;
            }
        showDigit(disp[digit-1]); //виводимо цифру з масиву disp у відповідний
розряд, враховуючи, що нумерація розрядів 1…3, а елементів у масиві 0…2
        //запалюємо десяткову крапку на другому розряді
        if(digit==2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
            else HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
        digit++; //збільшуємо номер розряду для засвічування при наступному
спрацюванні таймера
        //якщо номер розряду більший за кількість розрядів – починаємо
спочатку, з першого розряду
        if(digit>3) digit=1;
    }
    /* USER CODE END 1 */
```

Компілюємо код портованої програми, завантажуємо в мікроконтролер і запускаємо в режимі налагодження. Нагріваючи термістор теплом тіла, перевіряємо, чи регулятор вимикає вихід PA8 (світлодіод LED1) при досягненні верхньої межі зони гістерезису, й охолоджуючи, перевіряємо, чи вмикає регулятор вихід при зниженні температури до нижньої межі зони гістерезису.

4. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. В яких режимах може працювати модуль АЦП в STM32F072?
2. Яка функція використовується для зчитування результату аналогово-цифрового перетворення?
3. Яка функція очікує закінчення аналогово-цифрового перетворення?
4. Яким чином працює АЦП в режимі неперервного перетворення?
5. Чи можливо в програмі для мікроконтролера серії STM32F0 так само, як для AVR-мікроконтролера, змінювати стан цифрового виходу безпосереднім записом у регістр, а не викликом HAL-функції? Чому?
6. Чим програмно відрізняється організація обробки переривань мікроконтролерів серії STM32F0 з використанням HAL від ISR-стилю AVR-мікроконтролерів?

Завдання для самостійної роботи

1. У попередній програмі замість двопозиційного регулятора реалізувати пропорційний регулятор з імпульсним виходом: тривалість імпульсів у секундах розраховується $T_{імп}=k_п(pv-sp)$, де $k_п=2$

– коефіцієнт налаштування П-регулятора, pv – поточне значення регульованої величини, sp – завдання регулятора. Пауза між імпульсами – 3 секунди.

2. У програму регулювання з лабораторної роботи додати зміну завдання регулятора двома кнопками: кнопка Btn1 (вхід PC14) зменшує завдання на одну десяту градуса, кнопка Btn3 (вхід PC15) збільшує завдання на одну десяту, кнопка Btn2 (вхід PC13) підтверджує зміну завдання. Поки жодна з кнопок не тиснута, регулятор показує поточне значення регульованої величини. Натискання будь-якої кнопки призводить до переходу регулятора в режим зміни завдання: на дисплей виводиться значення завдання, а не поточне значення регульованої величини, причому якщо перехід в режим зміни завдання ініційований кнопкою Btn2 – виводиться завдання без змін, Btn1 – зменшене на одну десяту, Btn3 – збільшене на одну десяту. Повторне натискання кнопок Btn1 чи Btn3 відповідно зменшує або збільшує завдання. При тривалому натисненні кнопки завдання змінюється на одну десяту кожні 0,4 секунди. Для застосування зміненого завдання користувач повинен натиснути кнопку Btn2 впродовж 2 секунд з моменту останнього натискання (відпускання) будь-якої кнопки. Якщо після збільшення/зменшення завдання користувач не натиснув жодної кнопки продовж 2 секунд, зміна завдання скасовується, мікроконтролер продовжує працювати з попереднім завданням. Регулятор повертається звичайного режиму і показує поточне значення регульованої величини.

3. У програму з попереднього завдання додати пришвидшення зміни завдання при тривалому натисненні кнопки Btn1 або Btn3. Початковий період 0,4 секунди застосовується, якщо сумарна тривалість утримання натиснутої кнопки менше 1,2 секунди, далі він зменшується за лінійною залежністю $T=-0,1\cdot(t-1,2)+0,4$ до мінімально можливого 20 мс.

4. В програмі з лабораторної роботи передбачити можливість швидко змінити тип світлодіодного індикатора (з спільним катодом, з спільним анодом) одним рядком програми. Використати директиви умовної компіляції.

3.3. Лабораторна робота 10. Застосування прямого доступу до пам’яті для отримання результату аналогово-цифрового перетворення

Мета роботи: навчитися використовувати контролер DMA для передачі даних між периферією та оперативною пам’яттю.

Теоретичні відомості

DMA, Direct Memory Access (прямий доступ до пам’яті) — окремий блок, підключений до різноманітної периферії та пам'яті, який може самостійно (без участі CPU) переміщати дані

- з регістрів периферії в оперативну пам'ять і назад,
- з периферії в периферію,
- з пам'яті в пам'ять.

Розглянемо передачу даних з периферії в оперативну пам’ять на прикладі зчитування результату декількох аналогово-цифрових перетворень у масив.

Запуск модуля АЦП з передачею результату за допомогою DMA здійснюється функцією

*HAL_ADC_Start_DMA(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length),*

де pData – вказівник на буфер даних,
Length – кількість даних.

Перед викликом функції повинен існувати масив pData, проголошений з ключовим словом volatile (без нього компілятор вважатиме вміст масиву незмінним, оскільки зміна даних в масиві відбувається без виконання обчислювальним ядром інструкцій запису в масив). Розмір масиву підбирається так, щоб вмістити потрібну кількість результатів опитування заданої кількості каналів. Наприклад, якщо потрібно опитати 3 канали по одному разу кожен, достатньо 3 елементів у масиві, а якщо 2 канали опитуються 16 раз підряд, то варто передбачити 32 елементи.

Також має бути дозволене переривання від DMA по закінченню передачі даних. Саме воно сигналізуватиме про отримання потрібної кількості результатів АЦП у масив і можливість перейти до їх обробки.

В нормальному режимі DMA закінчить роботу з модулем АЦП після запису в масив останнього результату. В циклічному (кільцевому) режмі DMA знову запустить АЦП і в масиві значення будуть перезаписані новими результатами аналогово-цифрового перетворення.

Для запису результату 12-бітного АЦП достатньо 16-бітної беззнакової цілочисельної змінної uint16_t, в той же час функція вимагає вказівник uint32_t*, тому, наприклад, для масиву uint16_t buffer[32] функція матиме вигляд

HAL_ADC_Start_DMA(&hadc1, (uint32_t)buffer, 32);*

План роботи

1. Налаштувати DMA для роботи в нормальному режимі.
2. Налаштувати DMA для роботи в циклічному режимі.

Порядок виконання роботи

1. Відкрити проект з попередньої лабораторної роботи. В STM32CubeIDE відкрити ioc-файл і доналаштувати АЦП для використання DMA в нормальному режимі (рис. 3.7). Для цього обрати в лівій частині групу Analog, вибрати ADC, перейти на вкладку «DMA Settings», натиснути кнопку «Add», в списку обрати ініціатора DMA-запиту «ADC», залишити напрямок «з периферії в пам'ять», задати нормальний режим DMA, інкремент адреси пам'яті (результат АЦП зчитується з регістра результату, що має сталу адресу, а записується в масив у пам'яті, кожен наступний елемент в якому має більшу адресу), розмір даних – півслова (16 біт).

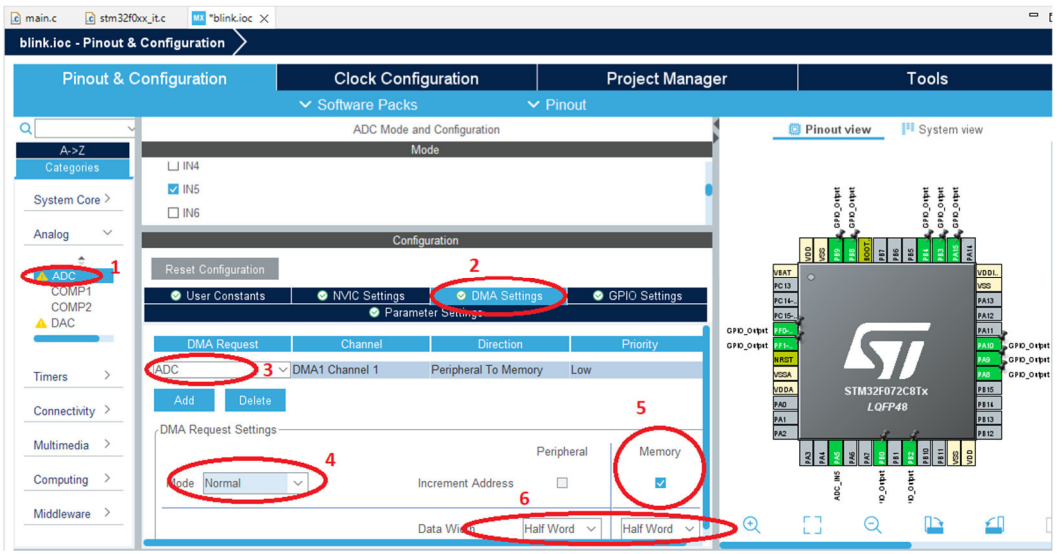


Рис. 3.7. Налаштування модуля АЦП для передачі результатів через DMA

Зберегти зміни й згенерувати оновлений програмний код проекту. В main.c повинна додатись функція ініціалізації DMA:

```
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
}
```

В файлі stm32f0xx_it.c з'явиться обробник переривання DMA1_Channel1_IRQHandler(), який викличе функцію XferCpltCallback(DMA_HandleTypeDef *hdma) по закінченню передачі. Відповідно в цю функцію перенесемо обчислення суми:

```
void XferCpltCallback(DMA_HandleTypeDef *hdma)
{
  adc_res=0;
  for (int i=0; i<32; i++) {
                    adc_res+=buffer[i];
    }
}
```

Альтернативою є розміщення коду одразу в обробнику переривання між /* USER CODE BEGIN DMA1_Channel1_IRQn 1 */ … /* USER CODE END DMA1_Channel1_IRQn 1 */ (деякі версії HAL не викликають XferCpltCallback). Тоді його вигляд стане наступним:

```
void DMA1_Channel1_IRQHandler(void)
{
  /* USER CODE BEGIN DMA1_Channel1_IRQn 0 */

  /* USER CODE END DMA1_Channel1_IRQn 0 */
  HAL_DMA_IRQHandler(&hdma_adc);
  /* USER CODE BEGIN DMA1_Channel1_IRQn 1 */
  adc_res=0;
   for (int i=0; i<32; i++) {
          adc_res+=buffer[i];
    }
  /* USER CODE END DMA1_Channel1_IRQn 1 */
}
```

Таким чином після закінчення обробки переривання по завершенню передачі DMA в змінній adc_res буде сума за 32 послідовні опитування аналогово-цифрового перетворювача.

Змінна adc_res і масив для збереження результатів buffer[32] проголошені в main.c, проте звертання до них здійснюється з обробника переривань з файлу stm32f0xx_it.c, тому в stm32f0xx_it.c їх потрібно проголосити з ключовим словом extern. Секція проголошення глобальних змінних матиме вигляд

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
uint8_t digit=1;
extern uint8_t disp[3];
extern volatile uint16_t buffer[32];
extern volatile uint32_t adc_res;
/* USER CODE END PV */
```

2. Модифікувати main.c з врахуванням того, що потреби 32 рази зчитувати результат АЦП вже немає, але потрібно синхронізувати початок обчислення температури з готовністю нового значення суми за 32 опитування АЦП. Оскільки в нормальному режимі DMA закінчує роботу після передачі останнього результату АЦП, його потрібно

запускати на початку кожної ітерації основного циклу while(1) викликом функції HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 32). Однократний запуск АЦП в режимі неперервного перетворення HAL_ADC_Start(&hadc) перед початком нескінченого циклу відповідно слід видалити або закоментувати.

Здачу синхронізації обчислень в обробнику переривання та в основному циклі класично можна вирішити проголошенням змінної-прапорця, яка встановлюватиметься з обробника переривання при готовності нового значення суми АЦП і слугуватиме умовою початку обчислення температури, регулювання та отримання масиву цифр для індикації. Коли свіжа сума результатів АЦП вже оброблена у основному циклі, прапорець скидається в нуль. Іншим варіантом може бути збереження попереднього значення змінної adc_res в змінній adc_res_old і порівняння на початку основного циклу цих змінних. Якщо в змінну adc_res у обробнику переривань буде записано нове значення суми результатів АЦП, в основному циклі умова if(adc_res!=adc_res_old) поверне істину і запуститься обчислення температури, регулювання та отримання масиву цифр для індикації. Останній варіант дозволяє не виконувати повторні обчислення температури і регулювання, якщо нове значення суми результатів АЦП співпадає з попереднім.

З врахуванням вищезазначеного, функція main() набуде вигляду:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC_Init();
  MX_TIM6_Init();
```

```
  /* USER CODE BEGIN 2 */
 //  HAL_ADC_Start(&hadc);
  HAL_TIM_Base_Start_IT(&htim6);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 32);
    if(adc_res!=adc_res_old) {
    temperature=calc_temperature(adc_res);
    adc_res_old=adc_res;
    if (temperature<sp-gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_SET);
    if (temperature>sp+gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_RESET);
    disp[0]=temperature/100;
    disp[1]=temperature/10-disp[0]*10;
    disp[2]=temperature%10;
    HAL_Delay(200);
    }
   /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Отримана програма після виклику функції HAL_ADC_Start_DMA() звільняється від потреби очікувати результат кожного перетворення, цей процесорний час може бути використаний для інших цілей. Готові результати АЦП зчитаються у обробнику переривання, буде обчислена сума, та якщо ця сума відрізнятиметься від попередньої, запуститься обчислення температури, позиційне регулювання та розбиття нового значення температури на розряди для індикації.

3. Скомпілювати проект, завантажити прошивку в мікроконтролер і налагодити програму.

Нами отримано більш ефективний код, який звільняє обчислювальне ядро мікроконтролера від потреби чекати результат АЦП і зчитувати його, але залишається потреба АЦП з DMA на початку кожного робочого циклу.

4. Змінити режим роботи DMA на кільцевий (рис. 3.8). Для цього обрати в лівій частині групу Analog, вибрати ADC, перейти на вкладку «DMA Settings», в списку обрати ініціатора DMA-запиту «ADC», залишити напрямок «з периферії в пам’ять», задати кільцевий режим

DMA, інкремент адреси пам’яті (результат АЦП зчитується з регістра результату, що має сталу адресу, а записується в масив у пам’яті, кожен наступний елемент в якому має більшу адресу), розмір даних – півслова (16 біт). Перейти на вкладку «Parameter Settings» і дозволити неперервні запити DMA, задавши «DMA Continuous Requests» значення «Enabled». Зберегти зміни й згенерувати оновлений програмний код проекту.

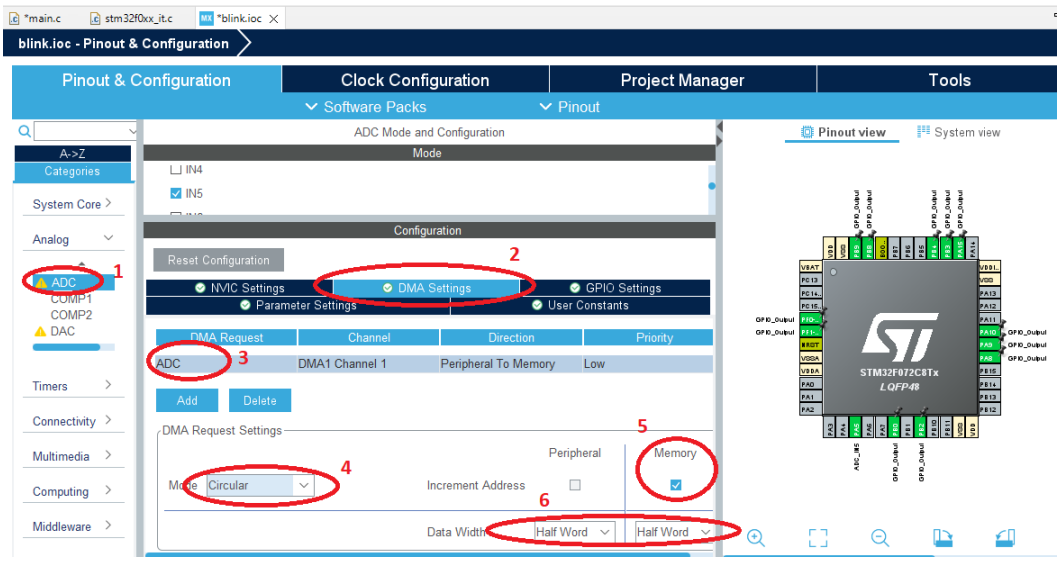


Рис. 3.8. Налаштування модуля АЦП для кільцевої передачі результатів через DMA

Функція ініціалізації АЦП набуде вигляду:

```
static void MX_ADC_Init(void)
{
  /* USER CODE BEGIN ADC_Init 0 */

  /* USER CODE END ADC_Init 0 */
  ADC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = ENABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = ENABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */

  /* USER CODE END ADC_Init 2 */
}
```

Тепер запущений одного разу АЦП з DMA буде працювати постійно. Коли останній результат АЦП із заданої кількості буде записано в масив у пам’яті, викличеться відповідний обробник переривання і обробить отриманий масив, а АЦП продовжить роботу, блок DMA записуватиме нові результати аналогово-цифрового перетворення спочатку виділеної під масив області оперативної пам’яті. Попередні результати АЦП у масиві буде перезаписано. Коли останній результат АЦП буде записано в кінець масиву в пам’яті, цикл повториться знову, DMA буде «по колу» записувати результати АЦП у масив у оперативній пам’яті.

З вищенаведеної причини виклик функції HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 8) можна винести з основного циклу й розмістити перед ним. Отже, остаточний запис функції main() виглядатиме:

```
int main(void)
{
```

```
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start_DMA(&hadc, (uint32_t*)buffer, 8);
  HAL_TIM_Base_Start_IT(&htim6);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(adc_res!=adc_res_old) {
    temperature=calc_temperature(adc_res);
    adc_res_old=adc_res;
    if (temperature<sp-gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_SET);
    if (temperature>sp+gist/2) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,
GPIO_PIN_RESET);
    disp[0]=temperature/100;
    disp[1]=temperature/10-disp[0]*10;
    disp[2]=temperature%10;
    HAL_Delay(200);
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

5. Скомпілювати проект, прошити мікроконтролер і налагодити програму. Тепер мікроконтролер виконує тіло основного циклу лише при істинності умови (adc_res!=adc_res_old), вільний процесорний час може використосуватись для виконання інших задач.

6. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. В яких режимах може працювати контролер DMA в STM32F072?

2. У яких випадках DMA може генерувати переривання?

3. У яких випадках доцільно використовувати контролер DMA?

4. Яка функція запускає модуль АЦП з передачею результату за допомогою DMA?

5. Який спосіб отримання результатів АЦП характеризується мінімальними затратами процесорного часу на зчитування 64 результатів?

Завдання для самостійної роботи

1. Продовжуючи шлях на збільшення ефективності використання процесорного часу, у програмі з лабораторної роботи замінити виклик функції затримки HAL_Delay(200) на формування затримки з використанням системного таймера, щоб надати змогу іншому коду виконуватись ці 200 мілісекунд.

2. Виміряти температуру напівпровідникового кристала мікроконтролера за допомогою вбудованого датчика температури. Вказати канал «Temperature sensor channel» в ioc-файлі під час створення проекту. Для перерахунку результату АЦП в одиниці температури ознайомитись з п. 13.8. Temperature sensor and internal reference voltage в технічному описі [3] та прикладом програмного коду в додатку A.7.15. Temperature configuration code example.

4. ПРАКТИЧНІ РОБОТИ

4.1. Практична робота 5. Використання таймера TIM1 для генерування трифазної системи напруг

Мета роботи: навчитись використовувати розширений таймер TIM1 мікроконтролера сімейства STM32 для генерування трифазної системи напруг.

Теоретичні відомості

Таймер TIM1 в мікроконтролерах STM32 – це таймер розширеного керування. Його можна використовувати для різноманітних цілей, у тому числі для вимірювання тривалості імпульсів вхідного сигналу (захоплення) або генерування вихідних сигналів (порівняння, ШІМ, комплементарний ШІМ із вставкою «мертвого» часу). Довжину імпульсу та період можна змінювати від кількох мікросекунд до кількох мілісекунд з використанням передподільника таймера. Коефіцієнт передподільника налаштовується регістром PSC: вхідна частота ділиться на число (1+PSC) й подається на вхід двійкового лічильника. Таймер має комплементарні виходи, що дозволяє одному каналу керувати одразу двома силовими ключами (один – підключення до позитивного полюса живлення, інший – до негативного). Трьох каналів ШІМ з комплементарними виходами достатньо, щоб керувати 6-ма ключами інвертора й генерувати трифазну напругу (рис. 4.1) [8].

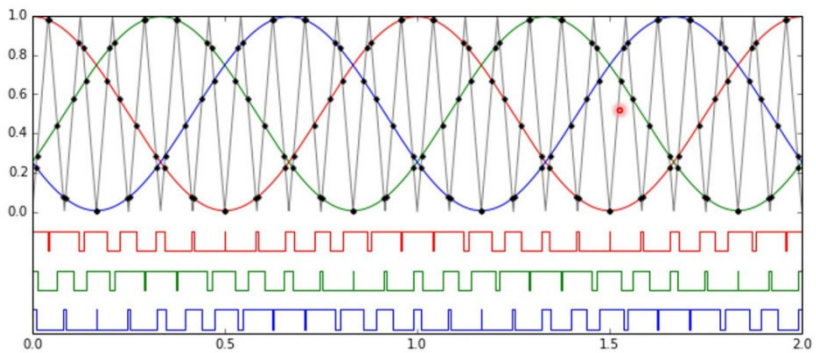


Рис. 4.1. Відповідність вмісту таймера (чорна лінія), синусоїд трьох фаз (вгорі) та ШІМ-напруги (внизу)

Якщо обрано режим рахунку вгору, вміст таймера збільшується до значення, вказаного в регістрі ARR, після чого таймер скидається в 0, а якщо режим рахунку вниз – зменшується до 0, після чого таймер завантажується вмістом регістра ARR. Отже, вміст регістра ARR і регістра передподільника PSC задає період імпульсів ШІМ-сигналу на

виході. Тривалість імпульсу задається в регістрах таймера CCR1, CCR2, CCR3.

Розрахуємо значення ARR та PSC для генерування ШІМ-сигналу частотою 5 кГц, що відповідає 100 імпульсам на 1 період синусоїди частотою 50 Гц. Нехай мікроконтролер тактується сигналом частотою 48 МГц, це відповідає періоду вхідного сигналу таймера 1/48000000 с. Таймер TIM1 16-бітний, а отже може рахувати максимум до 2^{16}=65536 перед тим, як переповниться. Таким чином, без застосування подільника вхідної частоти таймера (з подільником 1) він переповниться через проміжок часу 65536/48000000=0,001365333 с. Період ШІМ-сигналу частотою 5 кГц складає 0,0002 с. Отже, таймер повинен рахувати до числа, меншого 65536, а саме до 48000000/5000=9600.

У випадку отримання періоду ШІМ сигналу більшого, ніж період переповнення таймера без передподільника, потрібно було б збільшувати вміст регістра PSC (збільшувати коефіцієнт подільника), доки період переповнення таймера стане більшим або рівним бажаному періоду імпульсів.

План роботи

1. Налаштувати таймер TIM1 для генерування ШІМ-сигналу на 6 виходах мікроконтролера.

2. Запрограмувати мікроконтролер для генерування трифазної системи напруг з використанням DMA і TIM1.

3. Реалізувати зміну частоти вихідного сигналу за командою користувача.

Порядок виконання роботи

1. Створити в STM32CubeIDE новий проект і налаштувати таймер TIM1 наступним чином (рис. 4.2). Канали 1, 2, 3 налаштувати в режим генерування ШІМ-сигналу на пару комплементарних виходів (3 на рис. 4.2). Останні будуть відображені в у правій частині (4 та 5 на рис. 4.2). На вкладці «Parametr settings» у групі «Counter Settings» задати період ШІМ 9600, режим рахунку – вгору (Up), увімкнути функцію auto-reload preload. PSC залишити 0, що відповідатиме подільнику на 1.

Нижче в групі «PWM Generation Channel 1 and 1N» задати ненульове значення тривалості імпульсу (параметра «Pulse»). Це потрібно для спрощення налагодження: якщо світлодіод, підключений до відповідного каналу, світиться з стабільною яскравістю, значить таймер запустився у режимі генерування ШІМ-сигналу, проте не отримує нових значень тривалості імпульсу від блока DMA. Аналогічні налаштування виконати в групах «PWM Generation Channel 2 and 2N», «PWM Generation Channel 3 and 3N».

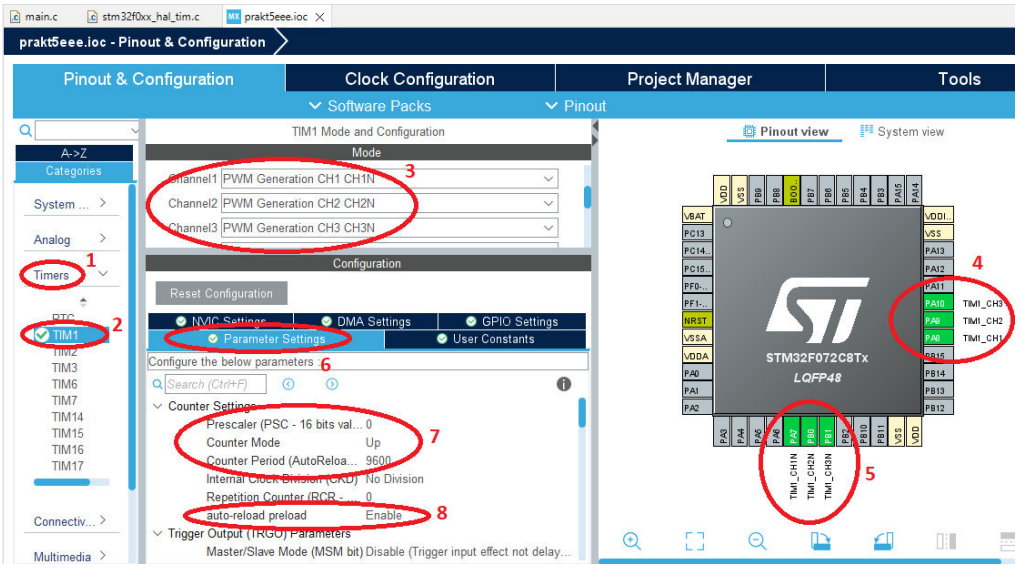


Рис. 4.2. Налаштування таймера TIM1 в графічному конфігураторі середовища STM32CubeIDE

Перейти на вкладку «DMA Settings», додати 3 запити (рис. 4.3) з напрямком «Memory To Peripheral» для циклічного запису тривалостей імпульсу з масиву, що містить значення синусоїди, в регістри тривалості імпульсу CCR1, CCR2, CCR3.

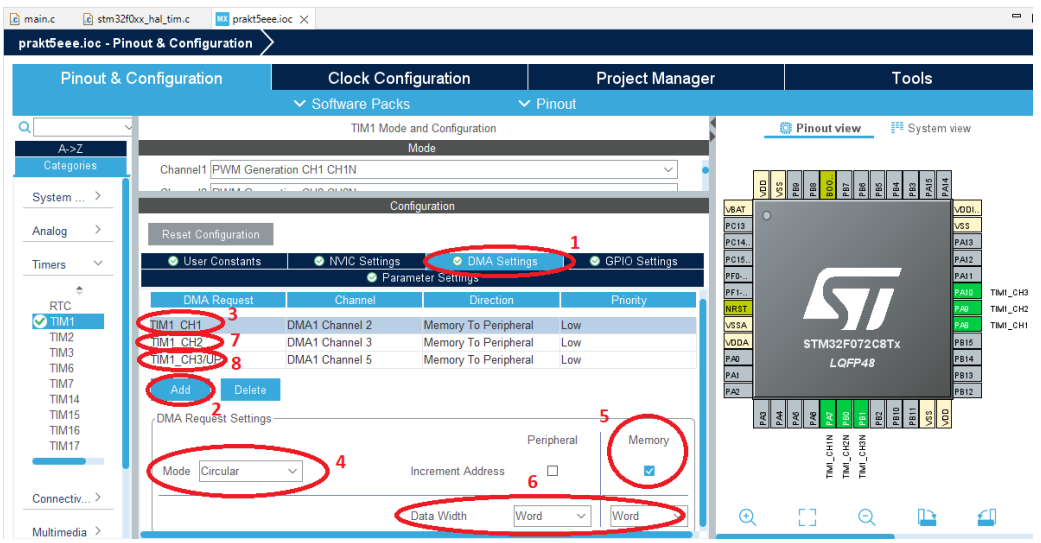


Рис. 4.3. Налаштування DMA таймера TIM1

Зверніть увагу: режим роботи (кільцевий), інкремент адреси й розмір даних потрібно задати для КОЖНОГО елемента в списку.

Зберегти зміни й згенерувати програмний код початкової ініціалізації. В результаті в main.c повинні з’явитись функція ініціалізації таймера TIM1 MX_TIM1_Init().

```
static void MX_TIM1_Init(void)
{
  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 0;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 9600;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 7000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) !=
HAL_OK)
  {
    Error_Handler();
  }
```

```
  sConfigOC.Pulse = 4000;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) !=
HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.Pulse = 2000;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) !=
HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput =
TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);
}
```

Також буде згенерована функція MX_DMA_Init():

```
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  /* DMA interrupt init */
  /* DMA1_Channel2_3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
  /* DMA1_Channel4_5_6_7_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_5_6_7_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA1_Channel4_5_6_7_IRQn);
}
```

2. Сформувати масиви значень регістрів CCR1, CCR2, CCR3 для трьох фаз так, щоб максимум синусоїди відповідав значенню в

регістрі ARR (9600), а мінімум – нулю. Розмір масивів – 100 елементів. Передбачити зсув фаз на 120°. Для цього в табличному процесорі створити стовпець із аргументами для функції синуса – числами від 0 до 2π з кроком π/50; в сусідньому стовпці ввести формулу =SIN(A1), в наступному =SIN(A1-2*PI()/3), в наступному =SIN(A1+2*PI()/3). Отримані значення синуса потрібно промасштабувати в діапазон від 0 до 9600 формулами відповідно =ROUND(B1*4800+4800;0), =ROUND(C1*4800+4800;0), =ROUND(D1*4800+4800;0). Оскільки регістри зберігають цілочисельні значення, застосовуємо округлення до цілого (0 знаків після коми). Результати розрахунку значень для запису в регістри тривалості імпульсів CCR1, CCR2, CCR3 наведені в додатку Д.

Для перевірки правильності розрахованих значень побудувати графіки зміни значень регістрів CCR1, CCR2, CCR3 залежно від порядкового номера. В результаті маємо отримати три синусоїди із зсувом фаз 120° (2π/3) в межах від 0 до 9600 (рис. 4.4).

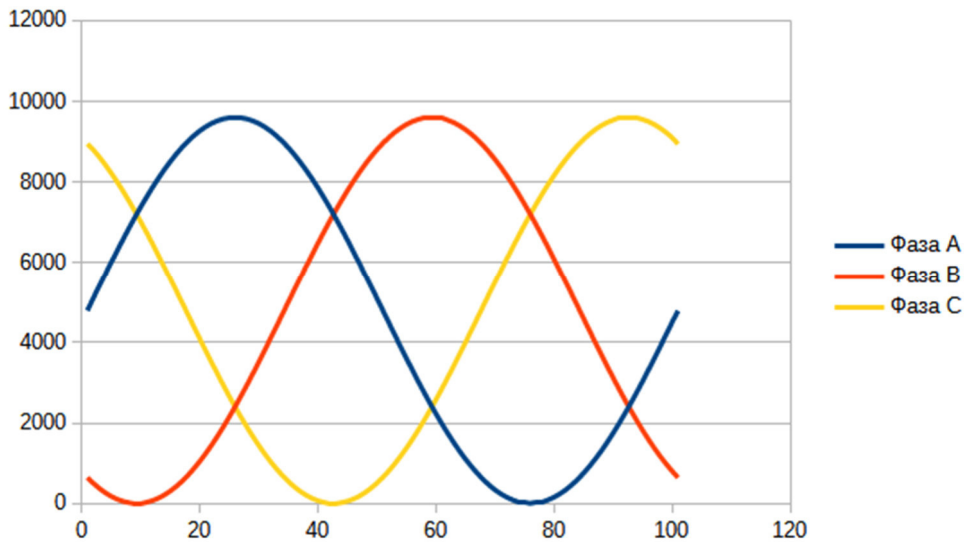


Рис. 4.4. Графіки значень регістрів CCR1, CCR2, CCR3 таймера TIM1

У файлі main.c у секцію

```
/* Private variables ---------------------------------------------*/
TIM_HandleTypeDef htim1;
DMA_HandleTypeDef hdma_tim1_ch1;
DMA_HandleTypeDef hdma_tim1_ch2;
DMA_HandleTypeDef hdma_tim1_ch3_up;
/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

додати проголошення масивів int tA[100]={…}, tB[100]={…}, tC[100]={…} з тривалостями імпульсів. Щоб спростити перенесення трьохсот

значень у текст програми, варто після кожного з стовпців з тривалостями імпульсів CCR1, CCR2, CCR3 додати стовпці з комою й екрануванням переходу на новий рядок (,\). Тоді пару стовпців «число-кома» можна скопіювати й вставити між фігурними дужками ініціалізації масиву без змін.

3. Запустити генерування ШІМ-сигналу таймером розширеного керування TIM1 в режимі DMA. Для цього використаємо функцію *HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length),* де *htim – вказівник на хендл таймера (в нашому випадку &htim1), Channel – канал таймера (від TIM_CHANNEL_1 до TIM_CHANNEL_3), *pData – вказівник на комірку пам'яті, починаючи з якої будуть зчитуватись значення тривалості імпульсу (відповідно від (uint32_t*)tA до (uint32_t*)tC), Length – довжина даних, які будуть передані з вказаної області пам'яті в регістри таймера. Відповідно в функції main() після ініціалізації периферії перед початком основного циклу у відповідну секцію додамо рядки:

```
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_1, (uint32_t*)tA, 100);
HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_2, (uint32_t*)tB, 100);
HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_3, (uint32_t*)tC, 100);
/* USER CODE END 2 */
```

Зверніть увагу: тіло основного циклу while(1) {…} залишилось пустим. Це означає, що після запуску таймера в режимі DMA копіювання даних у регістри тривалості імпульсів ШІМ-сигналу не потребуватиме жодних дій обчислювального ядра. Генерування ШІМ-сигналу з коефіцієнтом заповнення, що змінюється за законом синуса, для керування ключами трифазного інвертора здіснюватиметься апаратно, а всі обчислювальні ресурси мікроконтролера звільняються для виконання інших функцій.

4. Скомпілювати проект, завантажити прошивку в мікроконтролер та налагодити програму. Перевірити, чи змінюється сигнал на виходах, призначених для керування ключами фаз A, B, C.

Оскільки людина не здатна зафіксувати зміну сигналу 50 Гц на 3 виходах, в цілях налагодження допускається збільшити вміст регістра налаштування подільника PSC з 0 до 99, що призведе до зменшення частоти в 100 разів.

5. Передбачити в програмі можливість вибору частоти трифазної системи напруг 50 Гц або 25 Гц. Кнопка Btn3 (підключена до контакту PC15) призначена для встановлення частоти 50 Гц, кнопка Btn1 (підключена до контакту PC14) – 25 Гц.

Як і в лабораторній роботі 8, налаштуємо відповідні виводи мікроконтролера на вхід з підтяжкою до напруги живлення (рис. 4.5). Для кожного входу активувати підтяжку до напруги живлення (Pull-up).

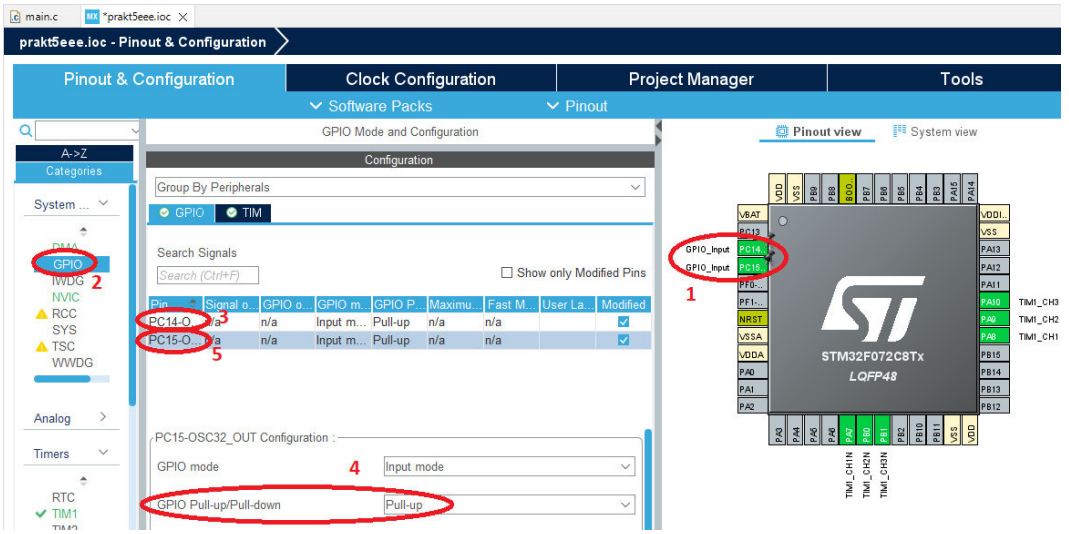


Рис. 4.5. Налаштування GPIO в STM32CubeIDE

Зберігаємо зміни й генеруємо оновлений програмний код. У програмі зміниться функція ініціалізації GPIO:

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pins : PC14 PC15 */
  GPIO_InitStruct.Pin = GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}
```

Додамо в основний цикл перевірку натискання кнопок. Щоб змінити частоту при натисканні кнопки, змінимо передподільник таймера, що спричинить зміну вихідної частоти. Це можна зробити, викликавши макрос __HAL_TIM_SET_PRESCALER(__HANDLE__, __PRESC__) з параметрами &htim1 (вказівник на хендл таймера TIM1) та 0 (для основної частоти 50 Гц) або 1 (для зниженої вдвічі частоти 25 Гц).

Отримаємо остаточний вигляд функції main():

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_1, (uint32_t*)tA, 100);
  HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_2, (uint32_t*)tB, 100);
  HAL_TIM_PWM_Start_DMA(&htim1, TIM_CHANNEL_3, (uint32_t*)tC, 100);
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_14)==GPIO_PIN_RESET) {
            __HAL_TIM_SET_PRESCALER(&htim1, 1);
     }
     if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_15)==GPIO_PIN_RESET) {
                    __HAL_TIM_SET_PRESCALER(&htim1, 0);
     }
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

6. Скомпілювати проект, прошити мікроконтролер і налагодити програму.

Завдання для самостійної роботи

1. Останню програму з практичної роботи модифікувати так, щоб частота ШІМ-сигналу не залежала від вибору частоти синусоїди 25/50

Гц, а залишалась рівною 5 кГц. Замість зміни коефіцієнта передподільника передбачити два окремих масиви для кожної фази (один з них відвічі більший), на використання яких переналаштовуватиметься DMA і TIM1 при натисканні кнопки.

2. В попередньому завданні реалізувати зміну частоти в момент переходу синусоїдної напруги фази А через нуль, а не безпосередньо в момент натискання кнопки.

3. Оцінити продуктивність обчислювального ядра мікроконтролера в обчисленні значень CCR1, CCR2, CCR3 таблиці з додатку Д: знайти, тривалості для скількох періодів синусоїди (кожен період – 100 значень одного регістра) може обчислити мікроконтролер за 10 секунд. Час перевіряти за допомогою функції HAL_GetTick(). Результат записати в глобальну змінну, значення якої переглянути в режимі налагодження (виведення результату на індикатор необов'язкове).

4. Написати програму для зміни частоти в межах 10…50 Гц кнопками «більше» (Btn3, PC15) і «менше» (Btn1, PC14). Для цього після кожного натискання кнопки перераховувати розмір і значення тривалостей імпульсів у масивах tA, tB, tC. Допускається на час перерахунку масивів встановлювати на виході нуль синусоїди (4800 у шкалі 0…9600).

5. У попередньому завданні забезпечити нерозривність синусоїди (заморожування скважності ШІМ-сигналу на виходах на час перерахунку не допускається) при зміні частоти. Передбачити два окремих масиви для кожної фази: один використовується для генерування сигналу з поточною частотою синусоїди, а інший заповнюється новими розрахованими значеннями тривалості імпульсу ШІМ-сигналу після натискання кнопки зміни частоти користувачем. Коли масив буде заповнено новими значеннями, DMA і TIM1 переналаштувати на використання нового масиву.

6. У програмі з практичної роботи додати затримку (dead-time) між вимкненням одного ключа фази й ввімкненням іншого, щоб убезпечити силові ключі від короткого замикання, що виникне якщо один напівпровідниковий ключ відкриється швидше, ніж інший встигне закритись. Вважати, що мікроконтролер формує ШІМ-сигнали на SLLIMM-модуль STGIPQ4C60T-HL (часові параметри перемикання вказані в таблиці 7 «Inductive load switching time and energy» технічного опису [9]).

7. Запустивши таймер в режимі лічильника зовнішніх імпульсів, що надходять від лічильника води з імпульсним виходом, реалізувати індикацію на дисплеї з лабораторної роботи 9 об'єму води, спожитої з моменту ввімкнення мікроконтролера. 1 імпульс відповідає 10 л води.

8. Написати програму для вимірювача довжини меблевої кромки/кабелю/іншої відрізної продукції. До кромки при перемотуванні на бобіну притиснуто ролик, на валу якого закріплено енкодер. Геометричні розміри ролика і розрядність енкодера такі, що 1000 імпульсам відповідає довжина 220 см. Довжину виводити на дисплей з лабораторної роботи 9.

4.2. Практична робота 6. Обмін даними за допомогою трансивера RS-485 та модуля USART

Мета роботи: навчитись використовувати модуль USART мікроконтролера для прийому та передачі даних в мережах RS-485.

Теоретичні відомості

Інтерфейс RS-485 – один з найпоширеніших інтерфейсів у промисловій автоматизації. Це напівдуплексний інтерфейс, відповідно в один момент часу дані можуть передаватись лише в одному напрямку. Для задання напрямку роботи (прийом або передача) трансиверів RS-485 багато з них мають окремі виводи RE (Receiver enable, ввімкнення приймача) і DE (Driver enable, ввімкнення передавача) (рис. 4.6).

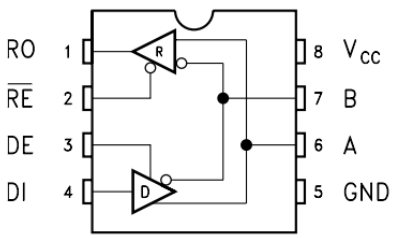


Рис. 4.6. Виводи трансивера ST485B

Оскільки активний рівень сигналів DE та RE протилежний, можна використовувати один вихід мікроконтролера для керування напрямком передачі: низький рівень активує приймач, а високий – передавач. Відповідно мікроконтролер під час передачі даних повинен утримувати логічну одиницю на цьому контакті, а решту часу утримувати логічний нуль, працюючи на прийом. Реалізувати це можна програмно, виставляючи лог. 1 перед заповненням буфера передачі й лог. 0 по події закінчення передачі, або використати контакт RTS апаратного керування потоком.

Регістр передачі модуля USART може зберігати лише одне двійкове слово для передачі (як правило, байт). Щоб повністю передати фрейм з декількох байт, необхідно завантажувати наступний байт у регістр передачі лише тоді, коли попередній байт залишить

регістр. Для цього можна: 1) чекати на звільнення регістра в основному циклі функцією HAL_UART_Transmit(); 2) використовувати переривання по спорожненню регістра передачі й в обробнику переривання записувати новий байт в регістр (функція HAL_UART_Transmit_IT); 3) використовувати контролер прямого доступу до пам'яті (DMA), який передаватиме черговий байт з оперативної пам'яті на запит USART при звільненні регістра передачі. З точки зору використання процесорного часу останній варіант найефективніший.

Регістр передачі модуля USART може зберігати лише одне двійкове слово. Для отримання фрейму з декількох байт потрібно встигати зчитувати отриманий байт з регістра прийому модуля USART до того, як надійде новий байт. Для цього можуть бути використані аналогічні функції HAL_UART_Receive(), HAL_UART_Receive_IT(), HAL_UART_Receive_DMA(). Проте в більшості протоколів розмір фрейму змінний, що ускладнює використання DMA, може бути використаний для отримання фіксованої кількості байт. Тому частіше використовується механізм переривань по прийому байту.

Крім апаратних модулів послідовного інтерфейсу USART для передачі даних, наприклад на ПК, може використовуватись USB. Значна кількість програм збору даних з мікропроцесорних плат розроблена для роботи з послідовним інтерфейсом, а не іншими класами USB-пристроїв. Тому за наявності USB-інтерфейсу між мікроконтролером і хостом часто гарним вибором є робота мікроконтролера в ролі пристрою USB CDC (Communications Devices Class), що реалізує віртуальний послідовний інтерфейс. Це дозволяє зберегти сумісність з програмами отримання даних з COM-портів, а також досить просто реалізується в STM32CubeIDE.

План роботи

1. Запрограмувати мікроконтролер для обміну даними через інтерфейс RS-485 з використанням модуля USART і трансивера SN75176.

2. Реалізувати передачу даних на комп'ютер з використанням USB CDC.

Порядок виконання роботи

1. Створити новий проект в середовищі STM32CubeIDE. Налаштувати модуль USART на швидкість передачі 115200 біт/с, 8 біт даних, доповнення до парності, один стоп-біт в асинхронному режимі (рис. 4.7). Налаштувати керування сигналами DE та RE трансивера інтерфейсу RS-485.

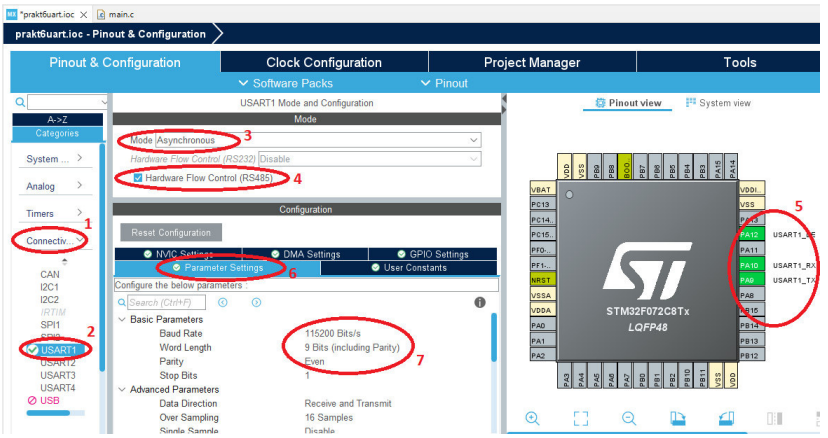


Рис. 4.7. Налаштування модуля USART для підключення до трансивера інтерфейсу RS-485

2. Активувати використання DMA для передачі (рис. 4.8). Для цього на вкладці «DMA Settings» додати канал DMA з пам’яті в периферію (запит USART1_TX).

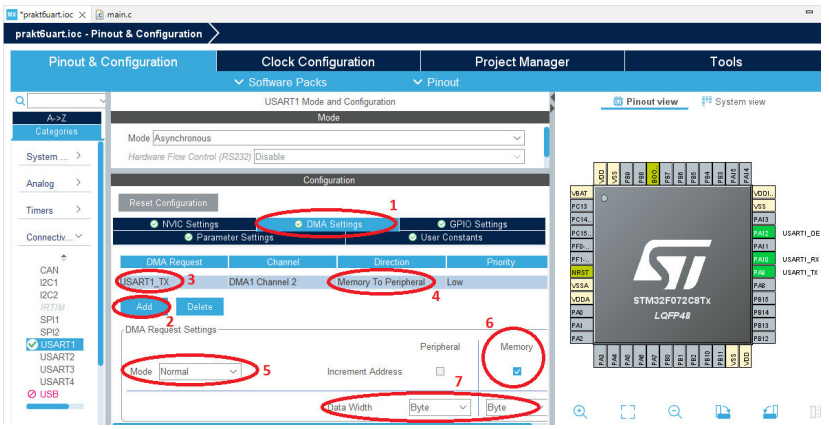


Рис. 4.8. Налаштування DMA для передачі даних в USART

3. Дозволити переривання від модуля USART для прийому. На вкладці «NVIC Settings» увімкнути переривання від модуля USART, встановивши чекбокс «USART1 global interrupt / USART1 wake-up interrupt through EXTI line 25».

4. Налаштувати GPIO PA8 на вихід, PC14 (Btn1) на вхід з підтяжкою до напруги живлення. Зберегти зміни в ioc-файлі й згенерувати програмний код.

У результаті отримаємо наступні функції ініціалізації периферії.

Ініціалізація модуля USART1:

```
static void MX_USART1_UART_Init(void)
{
  /* USER CODE BEGIN USART1_Init 0 */
```

```
  /* USER CODE END USART1_Init 0 */
  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_9B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_EVEN;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_RS485Ex_Init(&huart1, UART_DE_POLARITY_HIGH, 0, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */
}
```

Ініціалізація DMA:

```
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  /* DMA interrupt init */
  /* DMA1_Channel2_3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
}
```

Ініціалізація портів вводу-виводу загального призначення:

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pin : PC14 */
  GPIO_InitStruct.Pin = GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

5. Запрограмувати передачу кожні 5 секунд рядка «Test RS-485». Якщо натиснута кнопка Btn1, підключена до контакту PC14, замість вищевказаного рядка надсилати «button 1». Прийом відповіді здійснювати з використанням переривання.

Передбачимо буфери передачі trbuff[15], btnbuff[15] і прийому rcvbuff[15]. Для цього додамо відповідні рядки в секцію глобальних змінних

```
/* Private variables ---------------------------------------------*/
UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_tx;
/* USER CODE BEGIN PV */
char trbuff[15] = "Test RS-485\n";
char btnbuff[15] = "button 1\n";
volatile uint8_t rcvbuff[15];
/* USER CODE END PV */
```

Далі в основному циклі запустимо прийом даних викликом функції HAL_UART_Receive_IT(&huart1, rcvbuff, 15), де huart1 – хендл модуля USART1, rcvbuff – вказівник на початок області пам’яті для запису прийнятих байт, 15 – кількість байт, які очікується прийняти. Залежно від рівня напруги, що надходить з кнопки й зчитується функцією HAL_GPIO_ReadPin(), викликаємо HAL_UART_Transmit_DMA() або для передачі вмісту trbuff, або для передачі вмісту btnbuff.

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */
```

```
  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_UART_Receive_IT(&huart1, rcvbuff, 15);
    if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_14)==GPIO_PIN_RESET)
            HAL_UART_Transmit_DMA(&huart1, (uint8_t*)btnbuff, 15);
    else
            HAL_UART_Transmit_DMA(&huart1, (uint8_t*)trbuff, 15);
    HAL_Delay(5000);
   /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

6. В обробнику переривання по прийому передбачити перевірку першого байту отриманих даних. Якщо це код символа «b» – змінити стан виводу PA8 на протилежний.

Перейдемо до файлу stm32f0xx_it.c. Щоб обробник переривання мав доступ до масиву rcvbuff[15], пам'ять для якого виділена в main.c, додамо відповідний рядок в секцію глобальних змінних:

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
extern volatile uint8_t rcvbuff[15];
/* USER CODE END PV */
```

В обробнику переривання від модуля USART виконується перевірка джерела (події в модулі USART, яка спричинила переривання) й викликається відповідна функція зворотного виклику. У випадку прийому це функція *void*

*HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart).* Саме її повинен реалізувати користувач і додати в неї код, що виконуватиметься по події прийому байту. Додамо перевірку першого символу й зміну логічного сигналу на виході PA8 у секцію для коду користувача:

```
/* USER CODE BEGIN 1 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if(rcvbuff[0]=='b') HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
}
/* USER CODE END 1 */
```

7. Скомпілювати проект, завантажити програму в мікроконтролер і запустити в режимі налагодження. Підключити дві плати (свою та колеги) до мережі RS-485 контактами A, B, GND роз'єму RS485 у правій частині плати. Перевірити вміст символьних масивів при натиснутій та відпущеній кнопці (рис. 4.9).

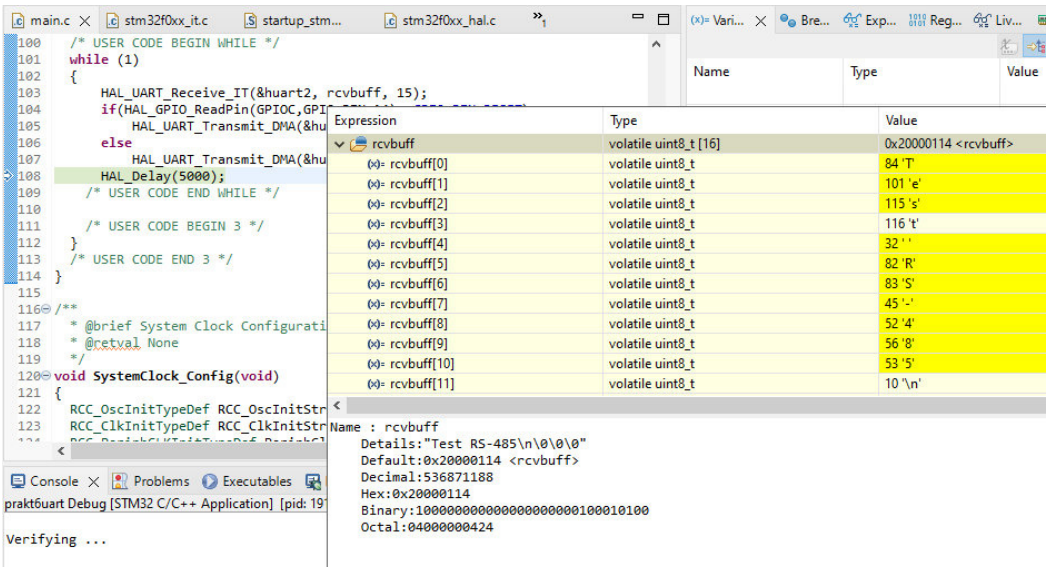


Рис. 4.9. Перевірка отриманих даних в режимі налагодження

При виконанні роботи наодинці можна використати перетворювач USB-RS485 та програму-монітор послідовного порту (послідовний термінал). Також можна використовувати вбудований послідовний термінал середовища Arduino IDE (рис. 4.10), обравши попередньо послідовний порт, присвоєний перетворювачу USB-RS485.

За відсутності перетворювача USB-RS485 або другої плати при самостійному виконанні роботи допускається замість USART1 використати USART2, контакти якого виведені на роз'єм ESP-01. Лівий верхній і правий нижній контакти роз'єму (TX, RX) з'єднати провідником. Це направить сигнал з виходу TX на вхід RX і дозволить перевірити правильність роботи програми при прийомі даних.

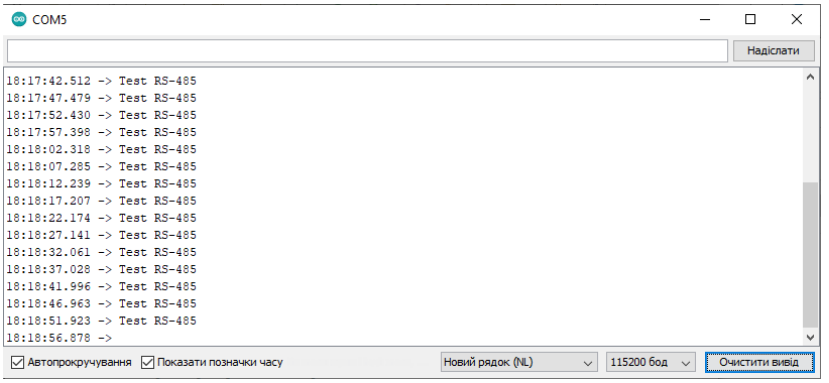


Рис. 4.10. Прийом даних від у Arduino IDE

8. Реалізувати аналогічну програму передачі даних на комп'ютер з використанням USB CDC Virtual COM, для чого спочатку створити ioc-файл.

Активувати функцію USB Full Speed пристрою (рис. 4.11) й використання вбудованої апаратної підтримки фізичного рівня інтерфейсу USB.

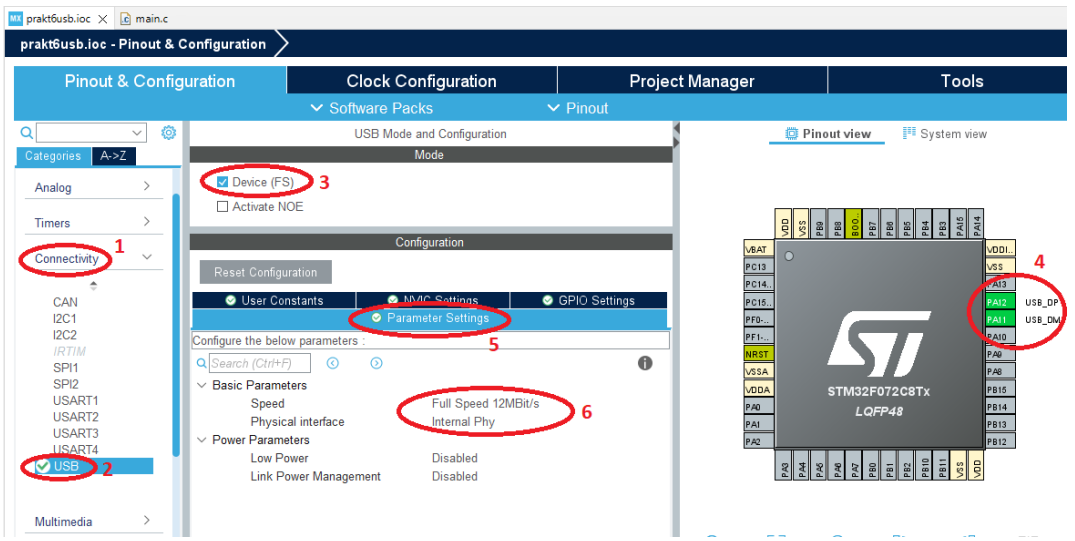


Рис. 4.11. Увімкнення USB-інтерфейсу

В розділі «Middleware» увімкнути включення у проект програмної реалізації класу USB CDC (Virtual Port Com), на вкладці дескрипторів пристрою залишити значення за замовчуванням (рис. 4.12). За потреби використовувати окремий драйвер для створюваного мікропроцесорного пристрою їх варто змінити на власні, адже саме за ними операційна система підбиратиме потрібний драйвер.

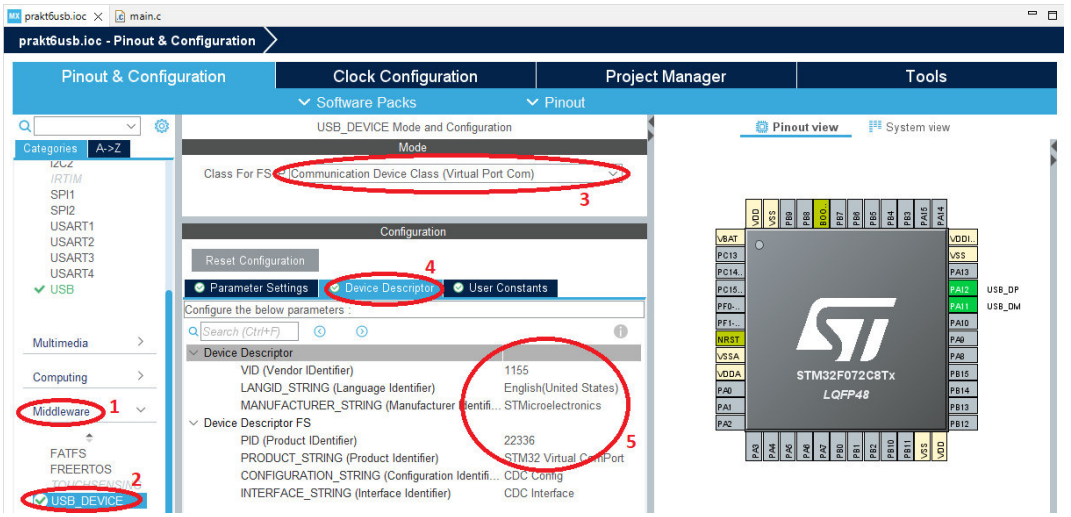


Рис. 4.12. Налаштування мікроконтролера як пристрою USB CDC

Також налаштувати контакт PA8 на вихід (GPIO_Output). Зберегти ioc-файл і згенерувати відповідний програмний код.

9. Додати передачу даних залежно від стану кнопки.

Додаємо ті самі масиви даних для передачі в main.c:

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
char trbuff[15] = "Test RS-485\n";
char btnbuff[15] = "button 1\n";
/* USER CODE END PV */
```

В функції main() замінюємо HAL-функції роботи з модулем USART функціями передачі

CDC_Transmit_FS(uint8_t Buf, uint16_t Len),*

де Buf – вказівник на початок масиву даних для передачі, Len – кількість байт.

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
```

```
  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_14)==GPIO_PIN_RESET)
            CDC_Transmit_FS((uint8_t*)btnbuff,15);
    else
            CDC_Transmit_FS((uint8_t*)buff,15);
    HAL_Delay(5000);
   /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

10. Додати обробку прийнятих даних аналогічно попередній програмі. В файлі USB_DEVICE/App/usbd_cdc_if.c знаходиться функція CDC_Receive_FS(), що викликається при отриманні нового масиву даних USB CDC. Додамо перевірку першого символа в отриманому повідомленні:

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 6 */
  if(Buf[0] == 'b') HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceFS);
  return (USBD_OK);
  /* USER CODE END 6 */
}
```

11. Скомпілювати проект, завантажити прошивку в мікроконтролер. Відключити програматор від комп'ютера та мікроконтролера. Підключити плату з мікроконтролером до USB-порта комп'ютера. Комп'ютер повинен розпізнати плату як послідовний порт з ідентифікатором виробника $0483_{(16)}$ та ідентифікатором продукту $5740_{(16)}$ і задіяти відповідний драйвер

(рис. 4.13). У випадку використання операційної системи на базі GNU/Linux dmesg виведе наступні повідомлення:

[248.857294] usb 1-3: New USB device found, idVendor=0483, idProduct=5740, bcdDevice= 2.00

[248.857315] usb 1-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3

[248.857323] usb 1-3: Product: STM32 Virtual ComPort

[248.857328] usb 1-3: Manufacturer: STMicroelectronics

[248.857334] usb 1-3: SerialNumber: 205E376E4234

[249.245575] cdc_acm 1-3:1.0: ttyACM0: USB ACM device

[249.245620] usbcore: registered new interface driver cdc_acm

[249.245623] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters

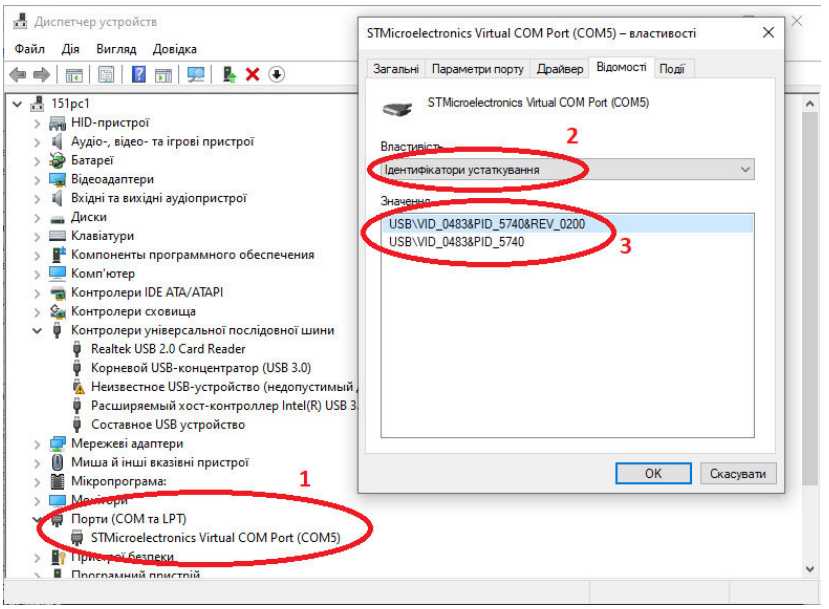


Рис. 4.13. Вигляд мікроконтролера STM32F072C8T6 у ролі пристрою USB CDC у диспетчері пристроїв

Відкрити монітор послідовного порту, обравши відповідний номер порта. В результаті повинні відобразитись ті ж повідомлення, що й при передачі через модуль USART (рис. 4.10).

СПИСОК ЛІТЕРАТУРИ

1. Основи проектування систем Інтернету речей. Периферія мікроконтролерів STM32 Конспект лекцій : навч. посіб. / уклад. : Ю. О. Оникієнко, А. Р. Рижова. ; КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2022. 127 с. URL: https://ela.kpi.ua/handle/123456789/48924 (дата звернення: 17.03.2023).

2. Noviello Carmine. Mastering STM32. Leanpub, 2018. (release 0.26). 792 с.

3. RM0091 Reference manual: STM32F0x1/STM32F0x2/STM32F0x8. URL: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1st m32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics. pdf (дата звернення: 25.04.2023).

4. AN2606. Application note. STM32 microcontroller system memory boot mode. URL: https://www.st.com/resource/en/application_note/an2606-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf (дата звернення: 15.05.2023).

5. Datasheet STM32F072x8 STM32F072xB. URL: https://www.st.com/resource/en/datasheet/stm32f072c8.pdf (дата звернення: 25.04.2023).

6. AN4899. Application note. STM32 microcontroller GPIO hardware settings and low-power consumption. URL: https://www.st.com/resource/en/application_note/an4899-stm32-microcontroller-gpio-hardware-settings-and-lowpower-consumption-stmicroelectronics.pdf (дата звернення: 05.05.2023).

7. Application note 1753. A Simple Thermistor Interface to an ADC. URL: https://www.analog.com/en/design-notes/a-simple-thermistor-interface-to-an-adc.html (дата звернення: 12.05.2023).

8. Custom Signal generation using PWM and DMA. URL: https://community.st.com/s/article/Custom-signal-generation-using-pwm-and-dma (дата звернення: 07.06.2023).

9. Datasheet STGIPQ4C60T-HL. URL: https://www.st.com/resource/en/datasheet/stgipq4c60t-hl.pdf (дата звернення: 07.06.2023).

ДОДАТКИ

Додаток А

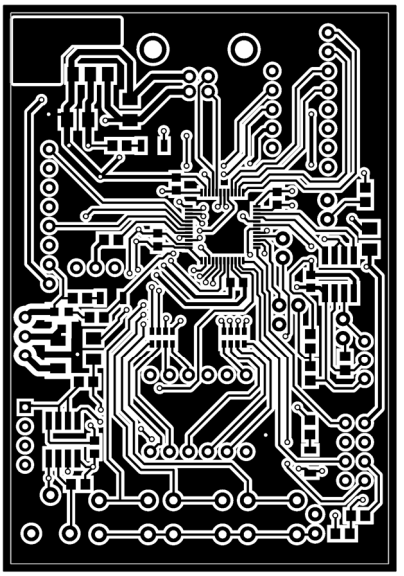
Список компонентів плати, що використовується для лабораторних робіт

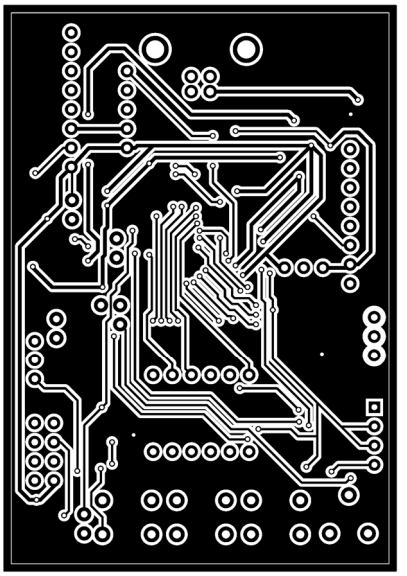
!PD	Штирьовий роз’єм, крок 0,1’’, тато, 2 контакти
ADC5_Sel	Штирьовий роз’єм, крок 0,1’’, мама, 2+1 контакти
AI	Штирьовий роз’єм, крок 0,1’’, мама, 2 контакти
AO	Штирьовий роз’єм, крок 0,1’’, мама, 2 контакти
BOOT0	Штирьовий роз’єм, крок 0,1’’, тато, 3 контакти
DO	Штирьовий роз’єм, крок 0,1’’, мама, 3 контакти
ESP-01	Штирьовий роз’єм 2-рядний, крок 0,1’’, мама, 4x2 контакти
GY-521	Штирьовий роз’єм, крок 0,1’’, мама, 8 контактів
Jmp120	Штирьовий роз’єм, крок 0,1’’, тато, 2 контакти
PA6	Штирьовий роз’єм, крок 0,1’’, мама, 2 контакти
PA7	Штирьовий роз’єм, крок 0,1’’, мама, 2 контакти
RC522	Штирьовий роз’єм, крок 0,1’’, мама, 8 контактів
RS485	Штирьовий роз’єм, крок 0,1’’, мама, 3 контакти
SWD	Штирьовий роз’єм, крок 0,1’’, тато, 5 контактів
USB	Роз’єм USB тип B, кутовий (Ninigi USBB-G)
Btn1	Кнопка тактова 6x6
Btn2	Кнопка тактова 6x6
Btn3	Кнопка тактова 6x6
C1	Танталовий конденсатор 10 мкФ, 10 В, корпус A (TAJA106K010RNJ)
C2	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C3	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C4	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C5	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C6	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C7	Керамічний конденсатор 100 нФ, 16 В, корпус 0603
C9	Танталовий конденсатор 10 мкФ, 10 В, корпус A (TAJA106K010RNJ)
C10	Танталовий конденсатор 10 мкФ, 10 В, корпус A (TAJA106K010RNJ)
C11	Танталовий конденсатор 10 мкФ, 10 В, корпус A (TAJA106K010RNJ)
D1	Діод швидкий, корпус MiniMELF/SOD-80 (LL4148 або 1N4148)

LD1	Світлодіодний семисегментний індикатор, спільний катод, 3-розрядний, 0,36” висота символа (напр., E30361-L-O-0-W)
LED1	Світлодіод SMD, корпус 1206
LED2	Світлодіод SMD, корпус 1206
LED3	Світлодіод SMD, корпус 1206
Q1	N-канальний польовий транзистор MOSFET AO3400, корпус SOT-23
R1	Резистор 2 кОм ±5%, корпус 0805
R2	Резистор 2 кОм ±5%, корпус 0805
R3	Резистор 2 кОм ±5%, корпус 0805
R4	Резистор 2 кОм ±5%, корпус 0805
R5	Резистор 430 Ом ±5%, корпус 0805
R6	Резисторна збірка 4x1 кОм, корпус 4x0603 (YC164-JR-071KL)
R7	Резисторна збірка 4x1 кОм, корпус 4x0603 (YC164-JR-071KL)
R8	Резистор 10 кОм ±5%, корпус 0805
R9	Резистор 120 Ом ±5%, корпус 0805
R10	Резистор 240 кОм ±5%, корпус 0805
R11	Резистор 1 кОм ±5%, корпус 0805
R12	Резистор 2 кОм ±5%, корпус 0805
R13	Резистор 10 кОм ±5%, корпус 0805
R14	Резистор 10 кОм ±5%, корпус 0805
Rp	Підстроювальний резистор 500 кОм, корпус CA9V або RM065
Rt	Термістор NTC 10 кОм, корпус 0603
U1	Мікроконтролер STMicroelectronics STM32F072C8T6, корпус LQFP-48
U2	Rail-to-Rail операційний підсилювач, корпус SOIC-8 (Texas Instruments LMV358)
U3	Драйвер диференціальної шини RS485/RS-422, корпус SOIC-8 (Texas Instruments SN75176)
U4	Лінійний стабілізатор 3,3 В, корпус SOT-223 (Advanced Monolithic Systems AMS1117-3.3)

Додаток Б

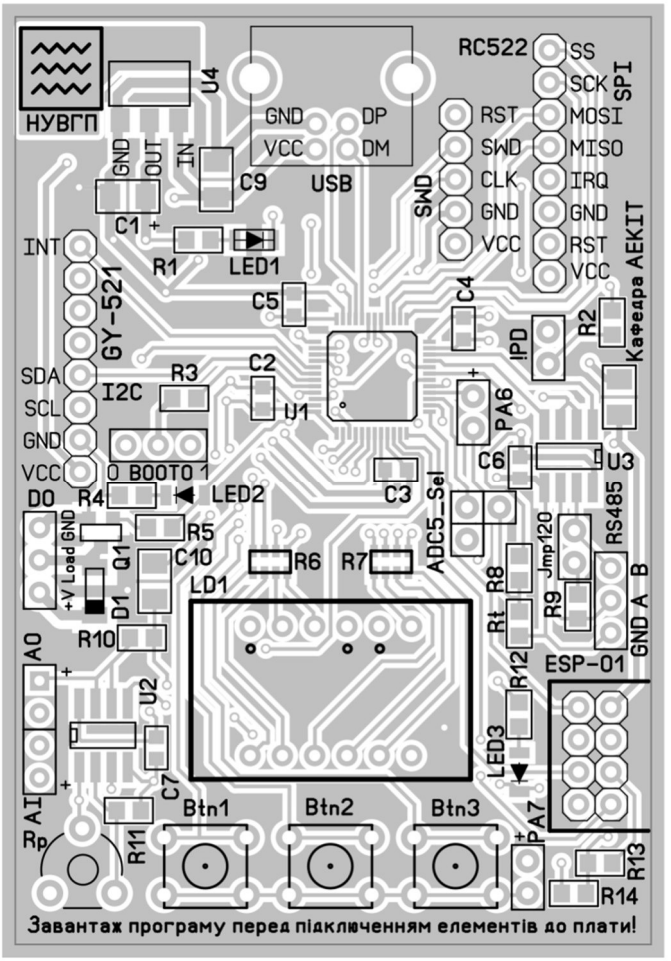
Верхній та нижній шари провідників друкованої плати 50x73, що використовується для виконання робіт (М 1:1)





Додаток В

Розташування електронних компонентів на платі
(шар шовкографії)



Додаток Ґ

Розрахунок таблиці відповідності між температурою термістора та сумою результатів АЦП

Для швидкого пошуку значення температури, яке відповідає сумарному результату АЦП, формують таблицю з наперед обчисленою відповідністю «температура – код АЦП». На основі таблиці температура-опір з документації термістора розраховується (знаючи опір термістора R при температурі T) напруга на виході подільника, утвореного термістором і постійним резистором, U. Оскільки й опорною напругою U_{ref}, й напругою живлення подільника є напруга живлення мікроконтролера 3,3 В, результат АЦП залежатиме від співвідношення напруг U/U_{ref}. Помноживши це співвідношення на максимально можливий результат АЦП (2^n-1) й кількість послідовних опитувань у циклі, що використовуються для усереднення, отримуємо сумарний результат ADC. Для оцінки розігріву термістора вимірювальним струмом розраховується його значення $I=U/(R+R_c)$ і теплова потужність ($P=UI$), яка виділиться на термісторі. Таблиця нижче містить результати розрахунку для термістора MF52A103F3950 з R_1=10 кОм (25° C) та $B_{25/100}$=3950 й резистора 7,5 кОм.

T,°C	R/R_1	R, кОм	U, В	I, мкА	P, мВт	U/U_{ref}	ADC
-25	14,43169	144,31694	3,13697	21,737	0,068	0,950598	124597
-23	12,70710	127,07098	3,11608	24,522	0,076	0,944267	123767
-21	11,21120	112,11205	3,09308	27,589	0,085	0,937297	122853
-19	9,91093	99,10927	3,06784	30,954	0,095	0,929650	121851
-17	8,77834	87,78338	3,04025	34,634	0,105	0,921287	120755
-15	7,78981	77,89811	3,01018	38,643	0,116	0,912176	119561
-13	6,92531	69,25312	2,97754	42,995	0,128	0,902284	118264
-11	6,16781	61,67814	2,94223	47,703	0,14	0,891584	116862
-9	5,50282	55,02816	2,90418	52,776	0,153	0,880054	115350
-7	4,91794	49,17941	2,86333	58,222	0,167	0,867677	113728
-5	4,40260	44,02605	2,81966	64,045	0,181	0,854443	111993
-3	3,94773	39,47734	2,77315	70,247	0,195	0,840349	110146
-1	3,54554	35,45538	2,72382	76,824	0,209	0,825400	108187
1	3,18931	31,89314	2,67172	83,771	0,224	0,809612	106117
3	2,87328	28,73284	2,61692	91,078	0,238	0,793005	103941
5	2,59246	25,92456	2,55953	98,73	0,253	0,775614	101661
7	2,34251	23,42514	2,49968	106,709	0,267	0,757479	99284
9	2,11971	21,19713	2,43754	114,994	0,28	0,738650	96816

11	1,92080	19,20804	2,37331	123,558	0,293	0,719186	94265
13	1,74296	17,42959	2,3072	132,373	0,305	0,699153	91639
15	1,58371	15,83715	2,23946	141,405	0,317	0,678624	88949
17	1,44092	14,40922	2,17034	150,622	0,327	0,657678	86203
19	1,31270	13,12701	2,10012	159,984	0,336	0,636399	83414
21	1,19741	11,97407	2,02908	169,456	0,344	0,614872	80593
23	1,09360	10,93595	1,95751	178,998	0,35	0,593186	77750
25	**1,00000**	**10,00000**	1,88571	188,571	0,356	0,571429	74898
27	0,91551	9,15506	1,81397	198,138	0,359	0,549686	72048
29	0,83913	8,39131	1,74255	207,661	0,362	0,528044	69212
31	0,77001	7,70010	1,67172	217,104	0,363	0,506582	66399
33	0,70738	7,07376	1,60174	226,434	0,363	0,485377	63619
35	0,65055	6,50553	1,53284	235,621	0,361	0,464497	60883
37	0,59894	5,98941	1,46523	244,636	0,358	0,444008	58197
39	0,55201	5,52008	1,39909	253,455	0,355	0,423967	55570
41	0,50928	5,09282	1,33459	262,054	0,35	0,404423	53008
43	0,47034	4,70342	1,27188	270,416	0,344	0,385418	50518
45	0,43481	4,34814	1,21106	278,525	0,337	0,366989	48102
47	0,40236	4,02364	1,15224	286,368	0,33	0,349164	45766
49	0,37269	3,72695	1,09548	293,936	0,322	0,331964	43511
51	0,34554	3,45540	1,04084	301,221	0,314	0,315406	41341
53	0,32066	3,20660	0,98834	308,221	0,305	0,299498	39256
55	0,29784	2,97844	0,93801	314,932	0,295	0,284244	37256
57	0,27690	2,76898	0,88983	321,356	0,286	0,269645	35343
59	0,25765	2,57652	0,84379	327,494	0,276	0,255695	33514
61	0,23995	2,39950	0,79987	333,35	0,267	0,242386	31770
63	0,22365	2,23653	0,75803	338,93	0,257	0,229705	30108
65	0,20864	2,08637	0,71821	344,239	0,247	0,217639	28526
67	0,19479	1,94789	0,68037	349,285	0,238	0,206172	27023
69	0,18201	1,82005	0,64444	354,075	0,228	0,195283	25596
71	0,17019	1,70195	0,61035	358,62	0,219	0,184955	24242
73	0,15927	1,59274	0,57805	362,927	0,21	0,175167	22959
75	0,14917	1,49168	0,54746	367,006	0,201	0,165896	21744
77	0,13981	1,39808	0,5185	370,867	0,192	0,157122	20594
79	0,13113	1,31132	0,49111	374,518	0,184	0,148822	19506
81	0,12308	1,23083	0,46522	377,971	0,176	0,140975	18478

83	0,11561	1,15610	0,44074	381,234	0,168	0,133559	17506
85	0,10867	1,08667	0,41763	384,317	0,161	0,126553	16588
87	0,10221	1,02211	0,39579	387,228	0,153	0,119937	15720
89	0,09620	0,96204	0,37517	389,977	0,146	0,113689	14901
91	0,09061	0,90610	0,35571	392,572	0,14	0,107791	14128
93	0,08540	0,85398	0,33734	395,021	0,133	0,102224	13399
95	0,08054	0,80537	0,32	397,333	0,127	0,096969	12710
97	0,07600	0,76001	0,30363	399,516	0,121	0,092010	12060
99	0,07176	0,71765	0,28819	401,575	0,116	0,087330	11446
101	0,06781	0,67806	0,27361	403,519	0,11	0,082912	10867
103	0,06410	0,64105	0,25985	405,353	0,105	0,078743	10321
105	0,06064	0,60642	0,24686	407,085	0,1	0,074807	9805
107	0,05740	0,57399	0,2346	408,72	0,096	0,071091	9318
109	0,05436	0,54361	0,22302	410,264	0,091	0,067583	8858
111	0,05151	0,51513	0,21209	411,721	0,087	0,064270	8424
113	0,04884	0,48841	0,20176	413,098	0,083	0,061140	8014
115	0,04633	0,46334	0,19201	414,399	0,08	0,058184	7626
117	0,04398	0,43979	0,18279	415,628	0,076	0,055390	7260
119	0,04177	0,41765	0,17407	416,79	0,073	0,052750	6914
121	0,03968	0,39684	0,16584	417,889	0,069	0,050253	6587
123	0,03773	0,37726	0,15805	418,927	0,066	0,047893	6277
125	0,03588	0,35883	0,15068	419,91	0,063	0,045660	5985

Додаток Д

Таблиця значень регістрів CCR1, CCR2, CCR3 таймера TIM1 для генерування трифазної системи напруг

№	x	sin(x)	sin(x-2π/3)	sin(x+2π/3)	CCR1	CCR2	CCR3
1	0,00000	0,00000	-0,86603	0,86603	4800	643	8957
2	0,06283	0,06279	-0,89571	0,83292	5101	501	8798
3	0,12566	0,12533	-0,92186	0,79653	5402	375	8623
4	0,18850	0,18738	-0,94438	0,75700	5699	267	8434
5	0,25133	0,24869	-0,96316	0,71447	5994	177	8229
6	0,31416	0,30902	-0,97815	0,66913	6283	105	8012
7	0,37699	0,36812	-0,98927	0,62115	6567	51	7782
8	0,43982	0,42578	-0,99649	0,57071	6844	17	7539
9	0,50265	0,48175	-0,99978	0,51803	7112	1	7287
10	0,56549	0,53583	-0,99912	0,46330	7372	4	7024
11	0,62832	0,58779	-0,99452	0,40674	7621	26	6752
12	0,69115	0,63742	-0,98600	0,34857	7860	67	6473
13	0,75398	0,68455	-0,97358	0,28903	8086	127	6187
14	0,81681	0,72897	-0,95732	0,22835	8299	205	5896
15	0,87965	0,77051	-0,93728	0,16677	8498	301	5600
16	0,94248	0,80902	-0,91355	0,10453	8683	415	5302
17	1,00531	0,84433	-0,88620	0,04188	8853	546	5001
18	1,06814	0,87631	-0,85536	-0,02094	9006	694	4699
19	1,13097	0,90483	-0,82115	-0,08368	9143	858	4398
20	1,19381	0,92978	-0,78369	-0,14608	9263	1038	4099
21	1,25664	0,95106	-0,74314	-0,20791	9365	1233	3802
22	1,31947	0,96858	-0,69966	-0,26892	9449	1442	3509
23	1,38230	0,98229	-0,65342	-0,32887	9515	1664	3221
24	1,44513	0,99211	-0,60460	-0,38752	9562	1898	2940
25	1,50796	0,99803	-0,55339	-0,44464	9591	2144	2666
26	1,57080	1,00000	-0,50000	-0,50000	9600	2400	2400
27	1,63363	0,99803	-0,44464	-0,55339	9591	2666	2144
28	1,69646	0,99211	-0,38752	-0,60460	9562	2940	1898
29	1,75929	0,98229	-0,32887	-0,65342	9515	3221	1664
30	1,82212	0,96858	-0,26892	-0,69966	9449	3509	1442
31	1,88496	0,95106	-0,20791	-0,74314	9365	3802	1233
32	1,94779	0,92978	-0,14608	-0,78369	9263	4099	1038
33	2,01062	0,90483	-0,08368	-0,82115	9143	4398	858

34	2,07345	0,87631	-0,02094	-0,85536	9006	4699	694
35	2,13628	0,84433	0,04188	-0,88620	8853	5001	546
36	2,19911	0,80902	0,10453	-0,91355	8683	5302	415
37	2,26195	0,77051	0,16677	-0,93728	8498	5600	301
38	2,32478	0,72897	0,22835	-0,95732	8299	5896	205
39	2,38761	0,68455	0,28903	-0,97358	8086	6187	127
40	2,45044	0,63742	0,34857	-0,98600	7860	6473	67
41	2,51327	0,58779	0,40674	-0,99452	7621	6752	26
42	2,57611	0,53583	0,46330	-0,99912	7372	7024	4
43	2,63894	0,48175	0,51803	-0,99978	7112	7287	1
44	2,70177	0,42578	0,57071	-0,99649	6844	7539	17
45	2,76460	0,36812	0,62115	-0,98927	6567	7782	51
46	2,82743	0,30902	0,66913	-0,97815	6283	8012	105
47	2,89027	0,24869	0,71447	-0,96316	5994	8229	177
48	2,95310	0,18738	0,75700	-0,94438	5699	8434	267
49	3,01593	0,12533	0,79653	-0,92186	5402	8623	375
50	3,07876	0,06279	0,83292	-0,89571	5101	8798	501
51	3,14159	0,00000	0,86603	-0,86603	4800	8957	643
52	3,20442	-0,06279	0,89571	-0,83292	4499	9099	802
53	3,26726	-0,12533	0,92186	-0,79653	4198	9225	977
54	3,33009	-0,18738	0,94438	-0,75700	3901	9333	1166
55	3,39292	-0,24869	0,96316	-0,71447	3606	9423	1371
56	3,45575	-0,30902	0,97815	-0,66913	3317	9495	1588
57	3,51858	-0,36812	0,98927	-0,62115	3033	9549	1818
58	3,58142	-0,42578	0,99649	-0,57071	2756	9583	2061
59	3,64425	-0,48175	0,99978	-0,51803	2488	9599	2313
60	3,70708	-0,53583	0,99912	-0,46330	2228	9596	2576
61	3,76991	-0,58779	0,99452	-0,40674	1979	9574	2848
62	3,83274	-0,63742	0,98600	-0,34857	1740	9533	3127
63	3,89557	-0,68455	0,97358	-0,28903	1514	9473	3413
64	3,95841	-0,72897	0,95732	-0,22835	1301	9395	3704
65	4,02124	-0,77051	0,93728	-0,16677	1102	9299	4000
66	4,08407	-0,80902	0,91355	-0,10453	917	9185	4298
67	4,14690	-0,84433	0,88620	-0,04188	747	9054	4599
68	4,20973	-0,87631	0,85536	0,02094	594	8906	4901
69	4,27257	-0,90483	0,82115	0,08368	457	8742	5202

70	4,33540	-0,92978	0,78369	0,14608	337	8562	5501
71	4,39823	-0,95106	0,74314	0,20791	235	8367	5798
72	4,46106	-0,96858	0,69966	0,26892	151	8158	6091
73	4,52389	-0,98229	0,65342	0,32887	85	7936	6379
74	4,58673	-0,99211	0,60460	0,38752	38	7702	6660
75	4,64956	-0,99803	0,55339	0,44464	9	7456	6934
76	4,71239	-1,00000	0,50000	0,50000	0	7200	7200
77	4,77522	-0,99803	0,44464	0,55339	9	6934	7456
78	4,83805	-0,99211	0,38752	0,60460	38	6660	7702
79	4,90088	-0,98229	0,32887	0,65342	85	6379	7936
80	4,96372	-0,96858	0,26892	0,69966	151	6091	8158
81	5,02655	-0,95106	0,20791	0,74314	235	5798	8367
82	5,08938	-0,92978	0,14608	0,78369	337	5501	8562
83	5,15221	-0,90483	0,08368	0,82115	457	5202	8742
84	5,21504	-0,87631	0,02094	0,85536	594	4901	8906
85	5,27788	-0,84433	-0,04188	0,88620	747	4599	9054
86	5,34071	-0,80902	-0,10453	0,91355	917	4298	9185
87	5,40354	-0,77051	-0,16677	0,93728	1102	4000	9299
88	5,46637	-0,72897	-0,22835	0,95732	1301	3704	9395
89	5,52920	-0,68455	-0,28903	0,97358	1514	3413	9473
90	5,59203	-0,63742	-0,34857	0,98600	1740	3127	9533
91	5,65487	-0,58779	-0,40674	0,99452	1979	2848	9574
92	5,71770	-0,53583	-0,46330	0,99912	2228	2576	9596
93	5,78053	-0,48175	-0,51803	0,99978	2488	2313	9599
94	5,84336	-0,42578	-0,57071	0,99649	2756	2061	9583
95	5,90619	-0,36812	-0,62115	0,98927	3033	1818	9549
96	5,96903	-0,30902	-0,66913	0,97815	3317	1588	9495
97	6,03186	-0,24869	-0,71447	0,96316	3606	1371	9423
98	6,09469	-0,18738	-0,75700	0,94438	3901	1166	9333
99	6,15752	-0,12533	-0,79653	0,92186	4198	977	9225
100	6,22035	-0,06279	-0,83292	0,89571	4499	802	9099

Навчальне видання

Реут Дмитро Тагірович

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ STM32 У STM32CUBEIDE. ПРАКТИКУМ

Навчальний посібник

Технічний редактор *Г.Ф. Сімчук*

Видавець і виготовлювач
Національний університет
водного господарства та природокористування,
вул. Соборна, 11, м. Рівне, 33028.

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції РВ № 31 від 26.04.2005 р.