Лабораторна робота №1

ЗНАЙОМСТВО З ПЛАТОЮ «STM32F411E-DISCOVERY» ТА СЕРЕДОВИЩЕМ STM32CUBEIDE. ЗАПИС ІНФОРМАЦІЇ НА ДИСКРЕТНИХ ЛІНІЯХ ПОРТІВ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

Мета: отримати базові навички роботи в програмному середовищі STM32CubeIDE. Ознайомитись з мікроархітектурою та послідовністю роботи з портами вводу/виводу загального призначення (GPIO) мікропроцесорної системи на базі ядра ARM Cortex-M4.

Теоретичні відомості

В рамках курсу розглядаються 32-бітний мікроконтролер сімейства STM32F4, що використовує ядро **ARM Cortex-M4** із підтримкою блоків обробки чисел з рухомою комою (FPU) та розширених цифрових сигналів (DSP). Процесори ARM є ключовим компонентом для великої кількості вбудованих систем (embedded systems). Процесори ARM широко використовуються в мобільних телефонах, планшетах та інших портативних пристроях, а з недавнього часу – і в ноутбуках. ARM засновані на RISC - архітектурі, що дозволяє зменшити споживання енергії процесором і, таким чином, робить їх гарним вибором для вбудованих систем.

Розробник може розглядати ядро ARM як набір функціональних блоків, з'єднаних різними шинами (рис. 1.1). Дані надходять в процесор через шину даних. Декодер (дешифратор) команд обробляє команди перед їх виконанням. ARM можуть працювати тільки з даними, які записані в регістрах, тому перед виконанням команд в регістри записуються дані. АЛП зчитує дані з регістрів, виконує необхідні операції і записує результат назад до спеціального регістру, звідки їх можна передати до зовнішньої пам'яті.

Процесори ARM містять до 18 регістрів: 16 регістрів даних і 2 регістра процесів. Всі регістри містять 32 біти та іменуються від R0 до R15. В залежності від контексту, ці регістри можуть використовуватися як регістри загального 5 призначення. Також є два програмних регістри, які називаються CPSR (Current Program Status Register) і SPSR (Saved Program Status Register), які використовуються для збереження поточного стану процесора і програми.

Таблиця 1.1. Основні характеристики **STM32F411VET6.**

Характеристика	**Значення**
Ширина слів даних, розрядів	32
Архітектура	Гарвардська
Набір інструкцій	RISC (з підтримкою Thumb-2)
Організація пам’яті програм, розрядів	32
Середній розмір команд, байт	2 (Thumb-2) або 4 (деякі команди)
Тип переривань	Векторизовані (NVIC)
Максимальна кількість пріоритетів переривань	256 рівнів
Затримка реагування на переривання	12 тактів (з PUSH всіх регістрів)
Підтримка FPU (Floating Point Unit)	Є (одинарна точність, IEEE 754)
Кеш команд	Є (опціонально, залежить від конкретної реалізації)

Підтримка DSP (Digital Signal Processing)	Є (SIMD, MAC, Saturation Arithmetic)
Налагоджувальний інтерфейс	JTAG, SWD (Serial Wire Debug)
Модулі трасування	ITM (Instrumentation Trace Macrocell), ETM (Embedded Trace Macrocell)
Кількість загальних регістрів	13 (R0-R12) + SP, LR, PC, xPSR
Моделі керування живленням	Sleep, Deep Sleep, Standby
Шина даних	AMBA (AHB-Lite, APB)
Максимальна тактова частота	До 100 МГц (STM32F411VET6)

Рис.1.1 – Загальний вигляд плати

Мікропроцесори даного типу побудовані на гарвардській архітектурі і мають внутрішню будову, яка мінімізує час виконання команд. У них використовуються внутрішні інтерфейси пам'яті з шириною шини більшою, ніж середня довжина команди. Це мінімізує число доступів до шини пам'яті, а, отже, і споживання електроенергії, пов'язане з операціями по шині і читанням пам'яті. STM32 – це платформа, в основі якої лежать мікроконтролери STMicroelectronics на базі ARM процесорів, різні модулі та периферія, а також програмні рішення (середовища розробки – IDE). Рішення на базі STM32 активно використовуються завдяки продуктивності мікроконтролера, його вдалій архітектурі, малому енергоспоживанню, невеликій вартості. У порівнянні з відомим проектом Arduino, контролери

STM32 виграють. Тактова частота мікроконтролерів Arduino нижче – 16 МГц проти 100 МГц у STM32. Кількість виводів та портів загального призначення у STM32 більше. Обсяг пам'яті у STM32 також вище. Не можна не відзначити pin-to-pin сумісність STM32 – для заміни одного виробу на інший не потрібно міняти плату. Недоліком STM32 можна назвати складну для вивчення архітектуру.

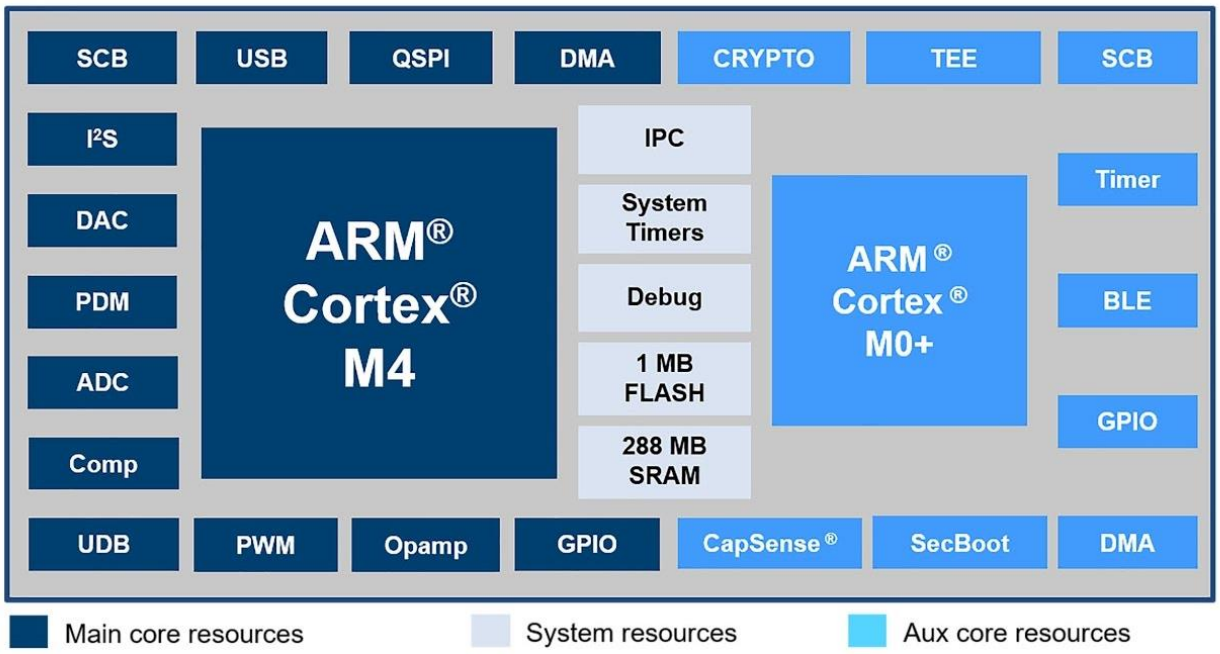

Рис.1.2 – Схема STM32F411VET6

Для більш детальної інформації по контролеру можна звернутись за посиланням: **https://www.st.com/en/evaluation-tools/32f411ediscovery.html#documentation**

STM32CubeIDE – це офіційне безкоштовне інтегроване середовище розробки від компанії STMicroelectronics. Воно дає можливість розробнику, який працює з мікроконтролерами STM32 , попередньо конфігурувати проект, налаштовувати периферію і частоти тактування мікропроцесора, генерувати код ініціалізації, проводити компіляцію і відлагодження програми та завантажувати її до пам’яті мікроконтролера. Дана платформа створена на базі об’єднання середовища програмування Atollic TrueStudio для STM32 та проекту STM32CubeMX, який дозволяє проводити попередні налаштування мікроконтролера за допомогою зручного графічного інтерфейсу.

Інтегрований STM32CubeMX дозволяє:

− здійснити вибір мікроконтролера STM32 та автоматично завантажити необхідні бібліотеки і драйвери;

− налаштувати тактування, периферію або додаткові опції;

− створити проект і згенерувати код ініціалізації.

STM32CubeIDE розроблена на базі ECLIPSE / CDT і включає в себе підтримку доповнень ECLIPSE, компілятор GNU C / C++ та налагоджувач, за допомогою якого можна аналізувати поточний стан мікроконтролера.

Порти – це пристрої введення / виведення, які дозволяють мікропроцесору передавати або приймати дані. Стандартний порт мікроконтролера STM32 має 16 розрядів даних, які можуть передаватися або прийматися паралельно. Кожному розряду (або біту) відповідає один вивід (ніжка) мікроконтролера. Ніжки мікроконтролера також називають пінами (pin). Для позначення імен портів використовуються латинські літери A, B, C тощо. Кількість портів введення / виведення змінюється в залежності від моделі мікроконтролера.

Порти GPIO (General purpose input-output, вхід / вихід загального призначення) розміщуються всередині кристалів мікропроцесорів. Вираз "загальне призначення" означає, що стан виводів встановлюється або зчитується програмно, тобто вивід не прив’язаний до певного конкретного апаратного вузла мікроконтролера. Порт GPIO обробляє як вхідні, так і вихідні цифрові сигнали. В якості вхідного порту його можна використовувати, наприклад, для зв'язку мікропроцесора з сигналами, отриманими від кнопок, або цифровими показаннями від датчиків. В якості вихідного порту його можна використовувати для формування сигналів управління зовнішніми операціями на основі виконуваних процесором програм. Наприклад, для управління увімкненням / вимкненням світлодіодного підсвічування дисплея, або передачі вихідних сигналів управління двигуном тощо.

У ранніх варіантах кожен порт був або виключно вхідним, або виключно вихідним. Однак зараз GPIO є «гнучким» у використанні своїх виводів (ніжок). Розробник може встановити їх призначення за власним розсудом, відповідно до наявних потреб.

В системі STM32 порти позначаються як GPIOA, GPIOB, GPIOC тощо. На схемі мікроконтролера відповідні портам виводи позначаються як PA0-PA15, PB0-PB15, PC0-PC15 тощо. Як вже було зазначено, порти 16-розрядні, тобто у кожного порту 16 виводів (ніжок). Режим і стан кожного виводу може бути встановлений окремо, незалежно від інших виводів. Так, кожен окремий вивід порту загального призначення може використовуватися в наступних режимах роботи:

− Input floating - вхід без підтягуючого резистора (вхід нікуди не підключений, «висить» в повітрі);

− Input pull-up - вхід з підтягуючим резистором, підключеним до живлення мікроконтролера. − Input pull-down - вхід з підтягуючим резистором, підключеним до загального проводу (землі).

− Analog - аналоговий вхід (вхід АЦП, компараторів тощо).

− Output open-drain - вихід з відкритим стоком. Функціонально аналогічний виходу з відкритим колектором. При низькому логічному рівні замикає вивід на землю, при високому - залишає не підключеним.

− Output push-pull - звичайний активний вихід. При низькому логічному рівні напруга на виводі дорівнює 0, при високому - напруга близька до напруги живлення мікроконтролера, зазвичай + 3 В.

− Alternate function push-pull - альтернативна функція виведення в звичайному (активному) режимі.

− Alternate function open-drain - альтернативна функція виведення в режимі відкритого стоку.

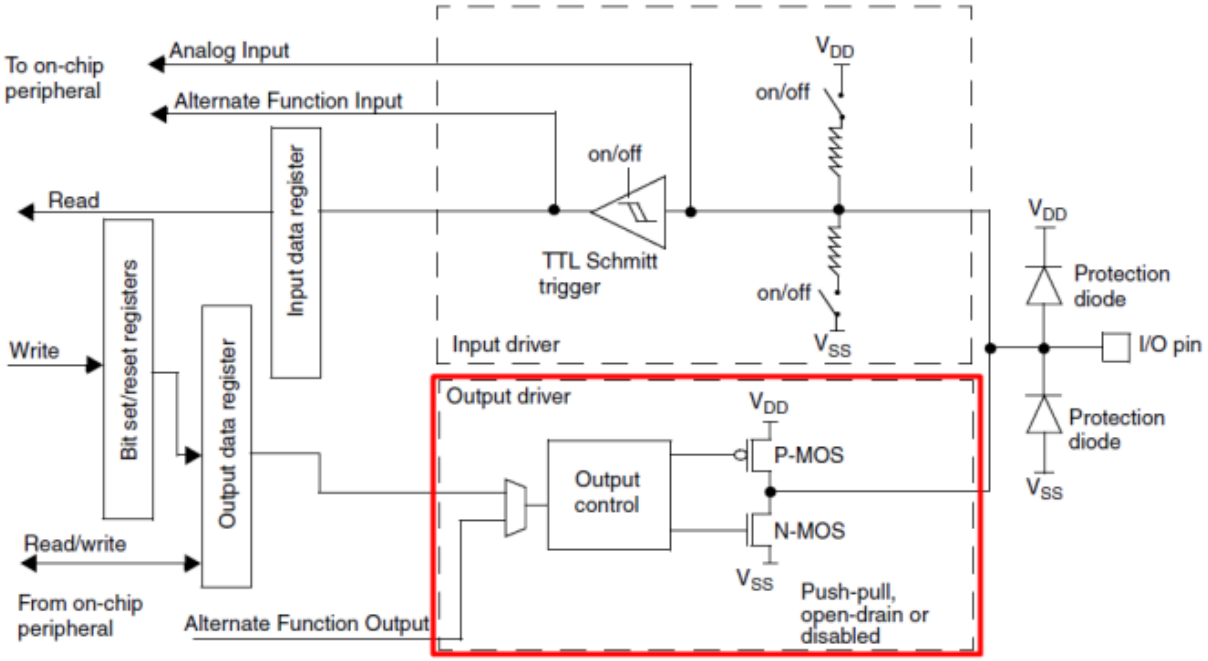

Рис.1.3 - Базова схема виводу загального призначення (виділено вихід)

Червоним кольором на схемі виділено вихідний драйвер (output driver) – частину схеми, яка відповідає за роботу виводу порту в режимі виходу. В такому режимі на виході порту працюють 2 комплементарних польових транзистори. В активному режимі (output push-pull) вони відкриваються по черзі, з'єднуючи вихід з шиною живлення або землею. У режимі з відкритим стоком (output opendrain) працює тільки нижній транзистор, замикаючи вихід на землю за низького логічного рівня.

Таблиця 1.2. Гранично-допустимі вихідні параметри портів

Параметр	Значення
Максимальних вихідний струм виводу	25 мА
Максимальний вхідний струм виводу	- 25 мА
Загальний сумарний вхідний та вихідний струм порту	Не більш ніж 150 мА.

УВАГА! Як видно з таблиці, до кожного виводу порту напряму можна підключати таке навантаження, щоб струм не перевищував 25 мА. Але за таких умов рівень логічного сигналу вийде за допустимі значення логічних рівнів. Щоб цього не сталося, **не можна навантажувати вивід струмом понад 8 мА.**

Хід виконання роботи

1.Виконати попередні налаштування плати «STM32 Blue pill» для керування вбудованим світлодіодом.

1.1. Запустити середовище програмування STM32CubeIDE.

1.2. У відповідь на отриманий запит обрати робочу директорію для зберігання проектів та натиснути кнопку Launch.

1.3. Створити новий проект, обравши в меню File –> New –> STM32 Project.

1.4. Обрати мікроконтролер, для якого буде вестися розробка. Для цього у полі пошуку вікна Target Selection ввести «STM32F411VET6» та натиснути на поле з відповідним

знайденим мікроконтролером у таблиці (рис. 1.4). Після успішного вибору плати натиснути кнопку Next.

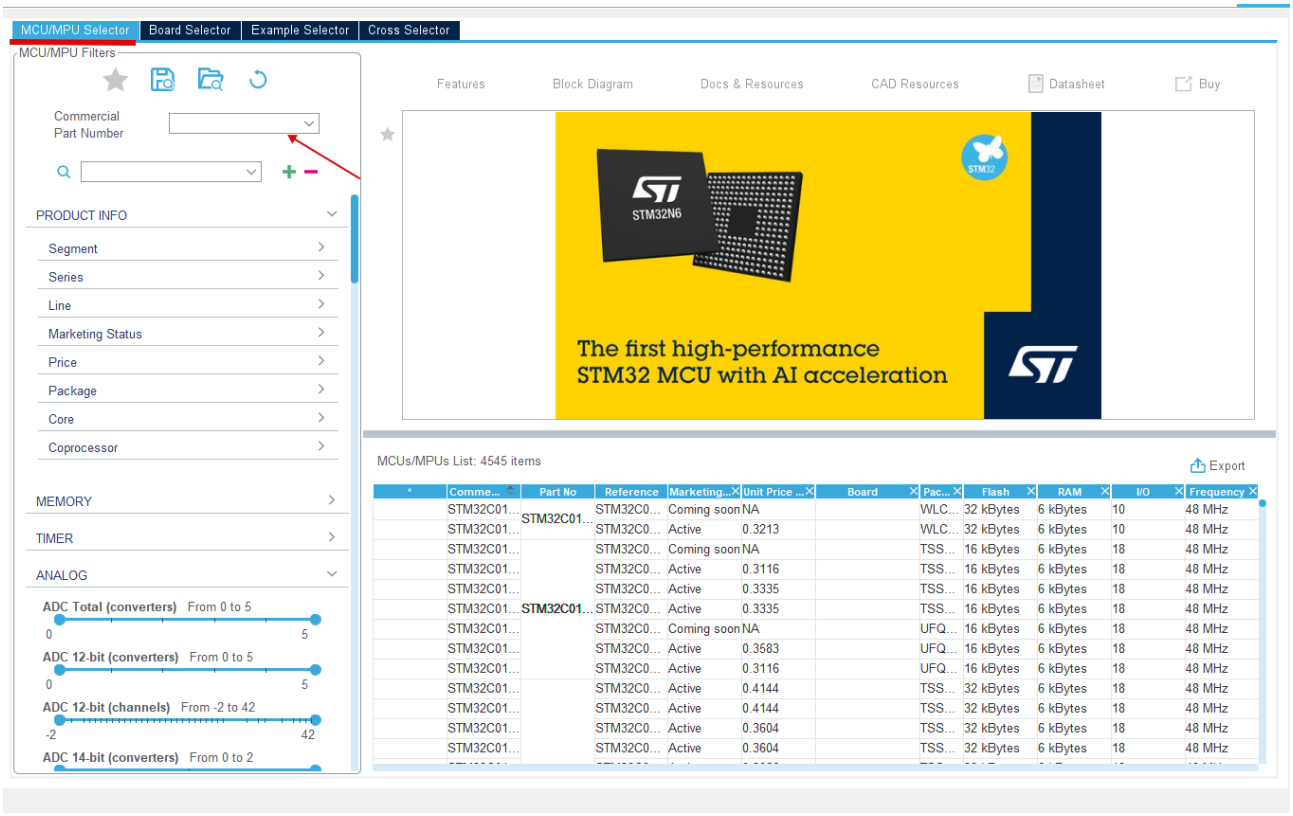

Рис. 1.4. Вибір плати

1.5. У вікні, яке з’явилося, ввести довільне ім’я проекту в полі Project Name, а також обрати мову програмування C в опції Target Language. Інші налаштування залишити за замовчуванням. Натиснути кнопку Finish.

1.6. На запит «Open Associated Perspective?» відповісти «Yes» для відкриття модуля STM32CubeMx.

1.7. Після проведення необхідних операцій буде відкрито головне вікно середовища STM32CubeIDE та модуль для конфігурації плати. В лівій частині вікна буде розміщене дерево файлів проекту (вкладка Project Explorer). На вкладці *Назва_проекту*.ioc показана графічна модель обраного мікроконтролера та розміщене меню для проведення відповідних попередніх налаштувань. Нижня частина головної робочої області містить вкладку Console для виведення поточної інформації. Загальний вигляд середовища STM32CubeIDE під час налаштування плати показано на рис. 1.5.

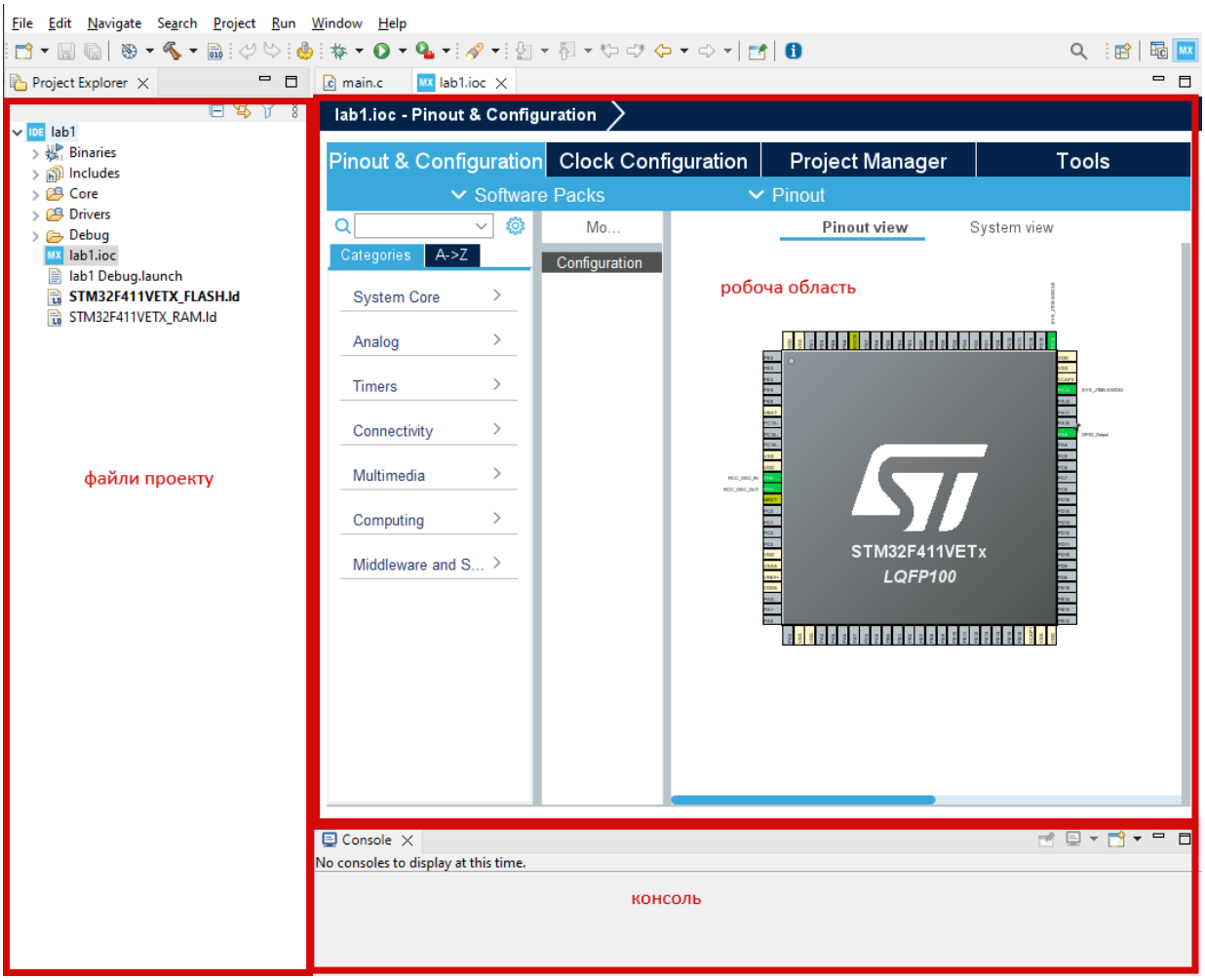

Рис. 1.5. Вікно STM32CubeIDE під час налаштування плати.

1.8. На вкладці Pinout & Configuration у підрозділі Categories обрати пункт підменю System Core –> SYS для налаштування режиму відлагодження. Для коректної роботи програматора, в опції Debug обрати інтерфейс «Serial wire» На графічній моделі мікроконтролера виводи PA13 та PA14 активуються та будуть позначені зеленим кольором.

1.9. Встановити режим тактування мікроконтролера від зовнішнього кварцового резонатора. Для цього у підменю RCC в опції High Speed Clock (HSE) обрати «Crystal/Ceramic resonator» . Активуються виводи PD0 та PD1.

1.10. Налаштувати вивід мікроконтролера, до якого підключено вбудований світлодіод на платі, для роботи в режимі виходу. Як видно зі документації, світлодіод підключено до виводу PA9. Щоб активувати вивід загального призначення, необхідно клацнути по ньому лівою кнопкою миші на графічній моделі контролера, та з випадаючого меню обрати необхідний режим роботи. Для встановлення режиму роботи виводу в якості виходу, обрати пункт «GPIO_Output» у випадаючому меню. Після активації вивід PC13 засвітиться зеленим кольором на графічний моделі мікроконтролера.

1.11. У підменю System Core –> GPIO налаштувати конфігурацію виводу PA9. Для цього на вкладці GPIO в таблиці обрати вивід PA9. Встановити за замовчуванням низький рівень на виході PA9, обравши в опції GPIO output level значення «Low».

1.12. Встановити режим активного виходу, обравши в опції GPIO mode значення «Output Push Pull».

1.13. За документацією відомо, що світлодіод апаратно підключено анодом до джерела живлення. Це означає, що він апаратно підтягнутий до високого рівня. Тому немає сенсу жорстко встановлювати режим підтягування програмно. Для цього в опції GPIO Pull-up/Pull-down обрати «No pull-up and no pull-down» - режим без підтягування.

1.14. В опції Maximum output speed встановити низьку швидкість роботи виводу, обравши значення «Low». Швидкість роботи виводу впливає на енергозбереження. Чим вища швидкість роботи, тим більше енергії споживає мікроконтролер.

1.15. Встановлені налаштування виводу PA9 повинні відповідати рис. 1.6.

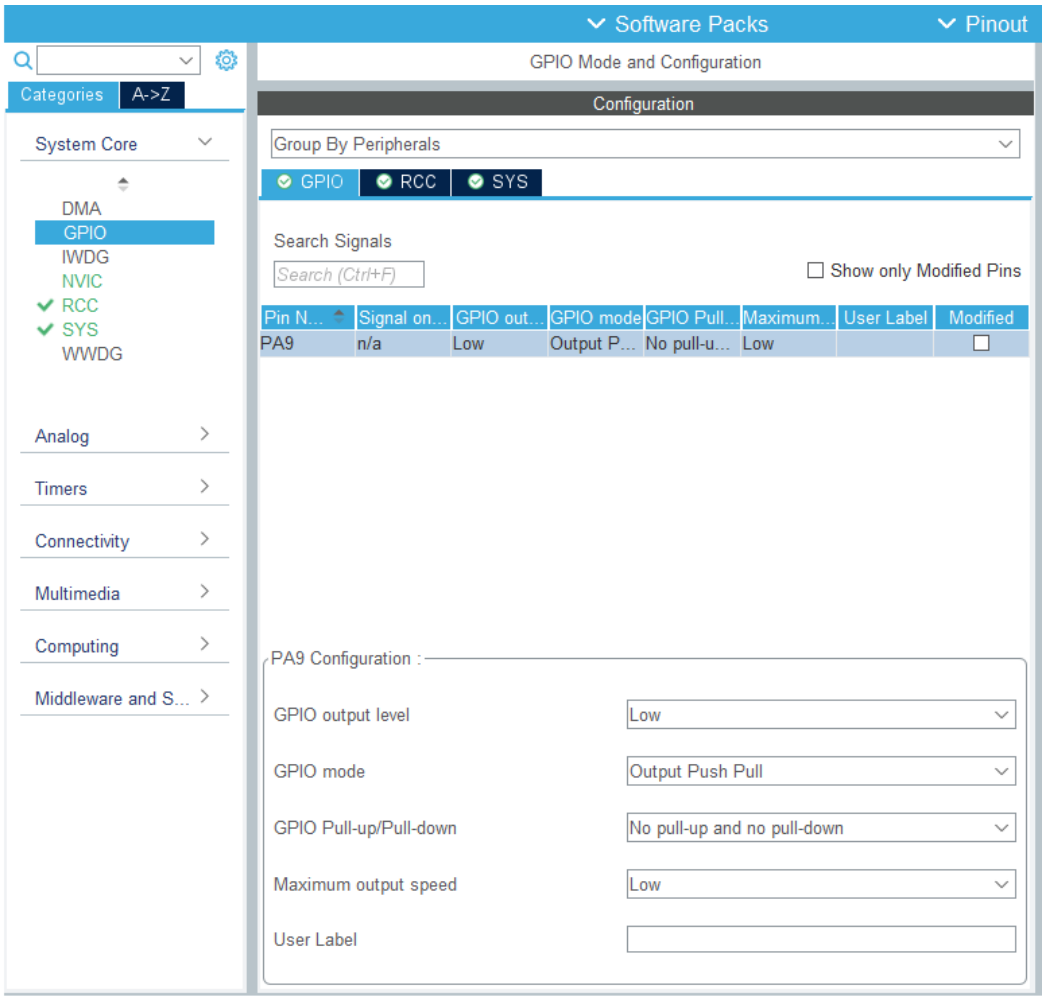

Рис. 1.6. Налаштування PA9.

1.16. На вкладці Clock Configuration основної робочої області встановити джерело тактування мікроконтролера від зовнішнього кварцового 17 резонатора і встановити максимальну частоту роботи мікроконтролера – 100 МГц, рис.1.7.

1.17. На вкладці Project Manager основної робочої області обрати підменю Code Generator та встановити в області Generated files прапорець біля опції «Keep User Code when re-generating» (не видаляти код користувача у випадку повторної генерації коду ініціалізації).

1.18. У підменю Advanced Settings обрати використання бібліотеки HAL для всіх доступних блоків.

1.19. Після завершення налаштувань, зберегти проект, натиснувши на піктограму (Save) або комбінацією клавіш Ctrl+S. На запитання, чи згенерувати код (Do you want generate code?), у спливаючому вікні відповісти «Yes» для автоматичної генерації шаблону програмного коду.

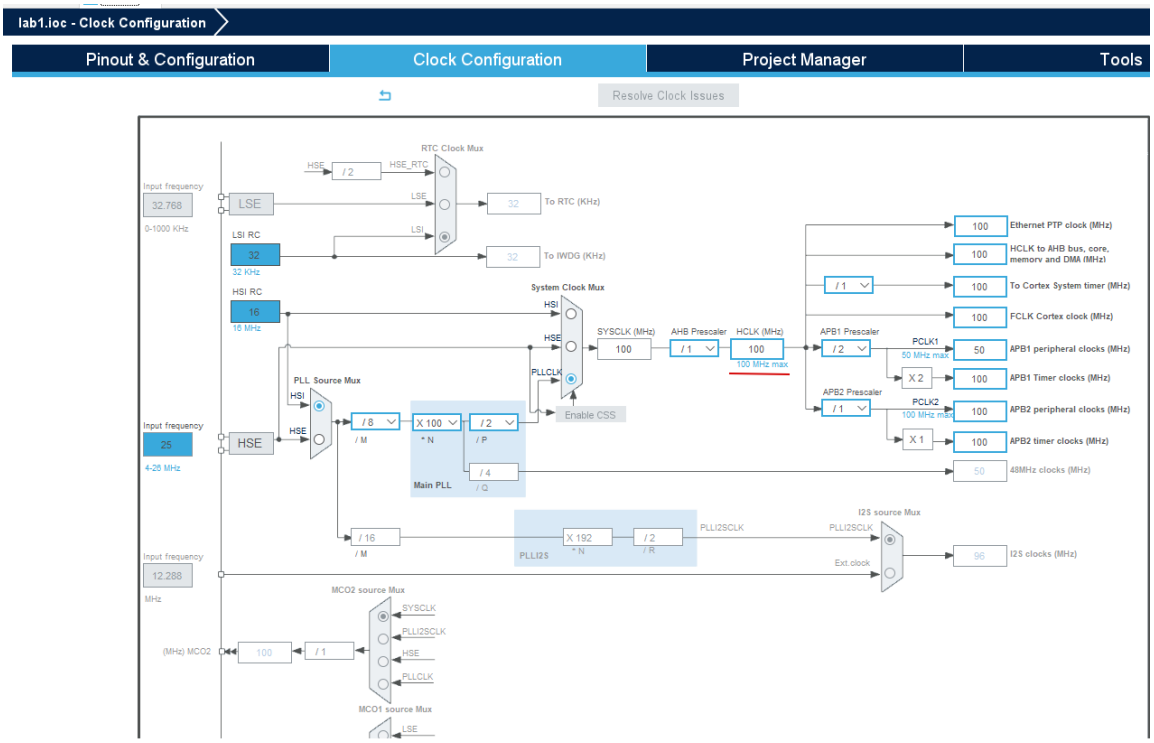

Рис. 1.7 – Налаштування тактування

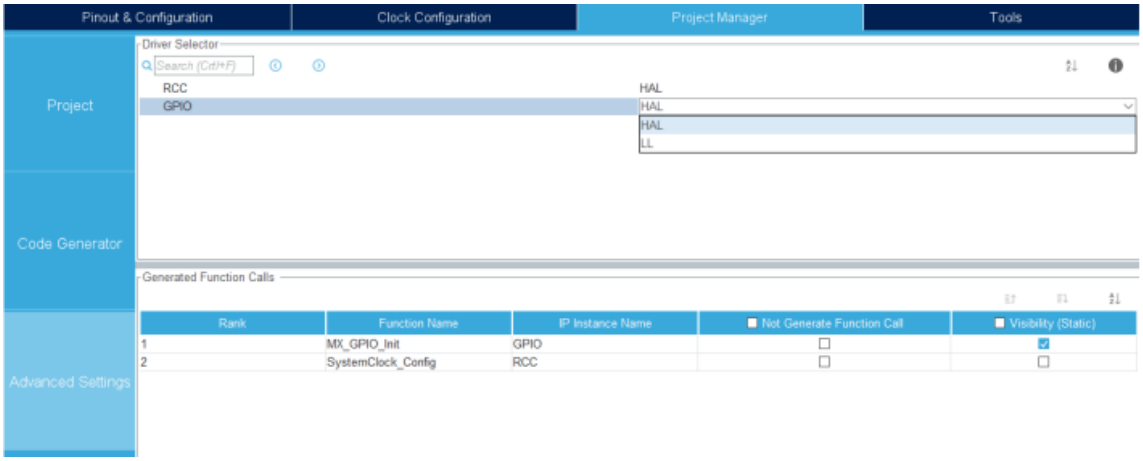

Рис.1.8 – Вибір бібліотеки HAL

2. Написати програму для керування вбудованим світлодіодом з використанням низькорівневої бібліотеки CMSIS.

2.1. В дереві структури проекту обрати основний файл для написання коду програми для роботи мікропроцесора. Він знаходиться за адресою: *Ім'я_проекту* –> Core –> Src –> main.c.

2.2. Самостійно ознайомитись зі структурою автоматично згенерованого файлу, який містить шаблон коду та виконані попередні налаштування мікроконтролера.

2.3. Вимкнути користувацький світлодіод на платі. Вимкнення необхідно встановити на виводі PA9 низький логічний рівень. За допомогою стандартної бібліотеки CMSIS це можна зробити, записавши «0» до 9-го біту регістра ODR (регістр вихідних даних) порту загального призначення A (вивід PA9 попередньо налаштовано для роботи у режимі виходу). Програмний код для виконання даної дії:

GPIOC->ODR |= 0 << 9;

Даний код необхідно розмістити всередині основного (нескінченного) циклу while(1). Саме цей цикл постійно виконує мікроконтролер під час своєї роботи, і в даному циклі завжди розміщується код основної програми. Середовище STM32CubeIDE автоматично

створило необхідний шаблон для циклу while(1). Отже, кінцевий варіант секції коду для реалізації вимкнення користувацького світлодіоду:

```
while (1)
{
    GPIOA->ODR |= 0 << 9;
}
```

Увага! Весь код користувача необхідно розміщувати всередині спеціально позначених секцій. Початок секції для коду користувача середовище STM32CubeIDE позначає як /* USER CODE BEGIN … */, а кінець секції - /* USER CODE END … */. Код користувача має розміщуватись всередині між цими написами. В іншому випадку, під час регенерації коду (у разі зміни налаштувань мікроконтролеру) код користувача буде стерто!

2.4. Провести компіляцію проекту для перевірки коректності написаного коду. Для цього натиснути на піктограму  або скористатися комбінацією клавіш Ctrl+B. Критерієм коректності коду буде відсутність помилок у консолі.

2.5. Підключити плату до ПК.

2.6. Завантажити написану програму до мікроконтролеру. Для цього натиснути на піктограму  (Run Debug). Під час першого запуску з'явиться вікно налаштувань Edit Configuration. В нашому випадку, ніяких змін вносити не потрібно, тому варто натиснути кнопку ОК та дочекатись завершення «прошивки» мікроконтролера – про це з'явиться відповідне повідомлення у консолі. Критерієм правильності роботи програми буде вимкнений користувацький світлодіод на платі.

2.7 Засвітити світлодіод на платі. Для цього необхідно встановити високий логічний рівень на виводі PA9, записавши «1» до 9-го біту регістра ODR порту загального призначення A. Із використанням бібліотеки CMSIS, код буде виглядати наступним чином:

```
while (1)
{
    GPIOA->ODR = 1 << 9;
}
```

2.8 Завантажити програму до мікроконтролеру. Критерієм правильності роботи програми буде увімкнений користувацький світлодіод на платі.

3. Написати програму для миготіння світлодіодом із використанням високорівневої бібліотеки HAL.

3.1. Для керування станом виводів в бібліотеці HAL існує функція HAL_GPIO_WritePin(). Інформація про функцію:

- Формат: void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState).
- Аргументи:
  - GPIOx – вибір порту (GPIOA, GPIOB, GPIOC …).
  - Pin – номер виводу (GPIO_PIN_0 … GPIO_PIN_15).
  - PinState – стан виводу:
    - GPIO_PIN_SET – високий рівень;
    - GPIO_PIN_RESET – низький рівень.

Таким чином, для встановлення високого рівня на виводі PA9 можна використати код:

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
```

Для встановлення низького рівня

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
```

3.2 Для організації затримки можна використати функцію HAL_Delay(). Інформація про функцію:

● Формат: void HAL_Delay (uint32_t Delay)

● Аргументи: o Delay – тривалість затримки в мілісекундах.

3.3. Використати зазначені функції для організації миготіння світлодіоду з затримкою 500 мс. Для цього написати наступний код:

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_SET);
HAL_Delay(500);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
HAL_Delay(500);
```

3.4. Завантажити програму до мікроконтролера. Критерієм правильності роботи коду буде миготіння користувацького світлодіоду з періодом 0,5 с.

4. Використати режим відлагодження Debug для відслідковування порядку виконання програми.

4.1. Натиснути на піктограму (Debug)  для запуску режиму відлагодження.

4.2. У вікні «Confirm Perspective Switch» натиснути кнопку Switch.

4.3. Користуючись інструментом для покрокового виконання програми (піктограма (Step Over) або клавіша F6), дійти до нескінченного циклу while(1). Після цього, у верхньому правому куті головної робочої області обрати вкладку SFRs для відслідковування стану регістрів периферійних блоків мікроконтролера.

4.4. Переглянути стан регістрів порту загального призначення A, обравши на вкладці SFRs пункт «GPIOA».

4.5. У випадаючому списку регістрів порту GPIOA обрати регістр ODR. Продовжуючи виконання програми у покроковому режимі, відслідкувати зміну стану 9-го біту регістра ODR (рис. 1.9). Порівняти значення 9-го біту регістру та стан світлодіода на платі.

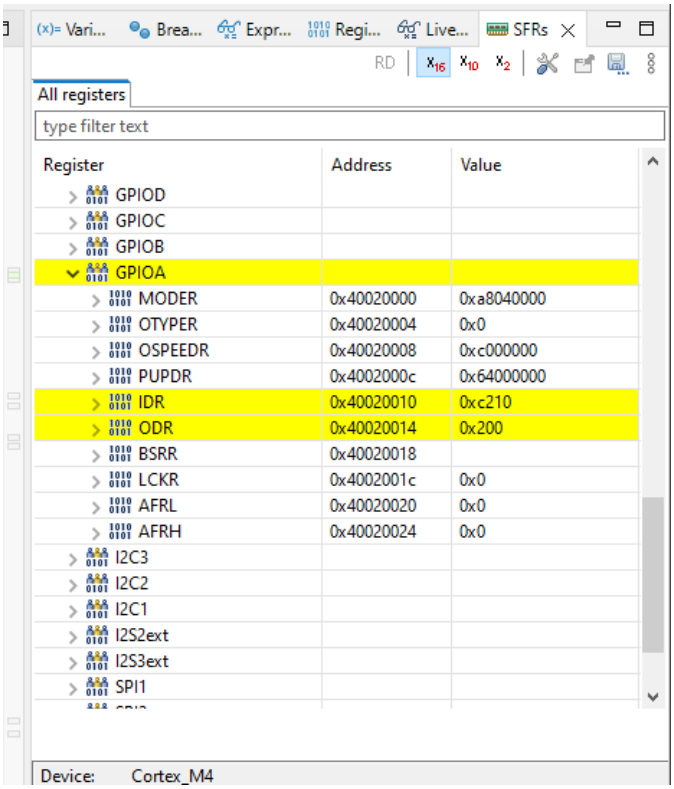

Рис.1.9 - Перегляд стану регістра ODR

4.6. Вийти з режиму відлагодження, натиснувши на піктограму (Terminate). 5. Переглянути алгоритм роботи функції HAL_GPIO_WritePin(). Для цього знайти цю функцію у коді програми, натиснути на її назву правою кнопкою миші та обрати пункт Open Declaration. Переглянути код функції та самостійно зробити висновки.

Індивідуальні завдання

1. Модифікувати програму таким чином, щоб світлодіод передавав прізвище студента азбукою Морзе. Переклад прізвища на азбуку Морзе дізнатись із відкритих джерел.

Зміст звіту

1. Тема та роботи;
2. Скріншот файлу .ioc з конфігурацією усіх задіяних пінів контролера;
3. Фрагмент коду з початку та до SystemClock_Config;
4. Висновки.

Контрольні запитання

1. Що таке порт мікропроцесора? Розкрийте особливості та структуру портів GPIO.

2. Назвіть мінімум 5 режимів роботи виводів портів GPIO.

3. Поясніть, яким чином налаштувати вивід мікроконтролеру для роботи в режимі активного виходу.

4. Опишіть логіку та поясніть код програми для миготіння світлодіодом із використанням бібліотеки стандарту CMSIS.

5. Опишіть логіку та поясніть код програми для миготіння світлодіодом із використанням бібліотеки HAL.

6. Поясніть алгоритм роботи функції HAL_GPIO_WritePin().

7.Назвіть призначення та способи роботи з регістром ODR портів GPIO.

Додаток 1

```c
#include "main.h"
#include <string.h>
#include <ctype.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
#define T_UNIT_MS 200
static void led_on(void)  { HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); }
static void led_off(void) { HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);}

static void delayT(uint32_t t_units)
{
    HAL_Delay(T_UNIT_MS * t_units);
}

typedef struct { char ch; const char* morse; } MorseMap;

static const MorseMap morse_table[] = {
    {'A', ".-"},    {'B', "-..."},  {'C', "-.-."},  {'D', "-.."},   {'E', "."},
    {'F', "..-."},  {'G', "--."},   {'H', "...."},  {'I', ".."},    {'J', ".---"},
    {'K', "-.-"},   {'L', ".-.."},  {'M', "--"},    {'N', "-."},    {'O', "---"},
    {'P', ".--."},  {'Q', "--.-"},  {'R', ".-."},   {'S', "..."},   {'T', "-"},
    {'U', "..-"},   {'V', "...-"},  {'W', ".--"},   {'X', "-..-"},  {'Y', "-.--"},
    {'Z', "--.."}
};

static const char* morse_lookup(char c)
{
    for (size_t i = 0; i < sizeof(morse_table)/sizeof(morse_table[0]); i++)
        if (morse_table[i].ch == c) return morse_table[i].morse;
    return NULL;
}

static void morse_send_symbol(char sym)
{

    led_on();
    if (sym == '.') delayT(1);
    else            delayT(3);
    led_off();
    delayT(1);
}

static void morse_send_char(char c)
{
    if (c == ' ')
    {
        led_off();
        delayT(7);
        return;
    }

    const char* code = morse_lookup(c);
    if (!code) return;
```

```
    for (size_t i = 0; code[i] != '\0'; i++)
        morse_send_symbol(code[i]);
    delayT(2);
}

static void morse_send_text(const char* text)
{
    for (size_t i = 0; text[i] != '\0'; i++)
    {
        char c = text[i];
        if (c >= 'a' && c <= 'z') c = (char)toupper((unsigned char)c);
        if (c == '_' || c == '-') c = ' ';
        morse_send_char(c);
    }

    delayT(7);
}
int main(void)
{


  HAL_Init();


  SystemClock_Config();


  MX_GPIO_Init();
  const char* surname = "SURNAME";
  while (1)
  {

        {
              morse_send_text(surname);
        }
  }

}
```