Практична робота №2

СТВОРЕННЯ ВЛАСНИХ БІБЛІОТЕК В СЕРЕДОВИЩІ STM32CUBEIDE

Мета: сформувати практичні навички створення власних та статичних бібліотек і підключення до STM32-проєкту в середовищі **STM32CubeIDE**.

Теоретичні відомості

Бібліотека в програмуванні — це набір зумовлених функцій, класів і ресурсів, які можна використовувати в додатку, щоб прискорити процес розробки. Такі набори містять код, який розробники можуть викликати зі свого власного додатка, щоб виконати певну задачу, наприклад, обробку зображень або роботу з мережею. Бібліотеки Вони дають змогу розробникаи значно скоротити час розробки, оскільки вони можуть використовувати готові компоненти у своїх проектах. Це також зменшує кількість помилок, які можуть виникнути при створенні нового коду, оскільки бібліотеки вже пройшли тестування і перевірку на помилки.

Бібліотеки можуть різнитися за своєю спеціалізацією, доступністю, метою та сферою використання. Наприклад:

Універсальні — це бібліотеки, які містять функції та класи для роботи з різними аспектами програмування і можуть використовуватися в різних галузях програмування.

Спеціалізовані — це бібліотеки, які розроблені для розв’язання конкретних завдань або використання в певній галузі програмування. Прикладами таких бібліотек можуть слугувати бібліотеки для роботи з графікою, машинним навчанням, роботою з базами даних тощо.

Внутрішні — це бібліотеки, які розробляються для використання всередині компанії або проєкту. Вони можуть містити унікальні рішення та функції, а також бути специфічними для конкретних потреб компанії або проєкту.

Зовнішні — це бібліотеки, які доступні для використання від сторонніх розробників. Вони можуть бути розроблені як відкрите програмне забезпечення або надаватися за плату.

Стандартні — це бібліотеки, які постачаються разом із мовою програмування і містять основні функції та класи для роботи з мовою. Вони можуть бути встановлені автоматично або вручну під час встановлення мови програмування.

Теоретично, можна написати будь-яку програму без використання бібліотек. Однак, їх використання значно спрощує і прискорює процес розробки програмного забезпечення. Вони надають готові рішення для найпоширеніших завдань у програмуванні, тим самим скорочуючи час і зусилля, які потрібно витратити на написання власного коду. Вони також дають змогу поліпшити якість програми, оскільки бібліотеки розробляються і тестуються професіоналами, що підвищує надійність і безпеку програмного забезпечення.

Хоча бібліотеки — корисний інструмент, їх використання може також призвести до низки проблем. Наприклад, якщо бібліотека містить помилку, то це може призвести до помилок у застосунку, які складно буде відстежити та виправити. Наведемо ще кілька проблем:

Конфлікти версій. Коли різні бібліотеки використовують різні версії однієї й тієї самої бібліотеки або залежності, це може призвести до конфліктів і помилок у роботі програми.

Несумісність. Деякі бібліотеки можуть бути несумісними з іншими наборами або з певною операційною системою. Це може призвести до помилок і збоїв у роботі програми.

Недостатня документація. Деякі бібліотеки можуть мати недостатню документацію, що може ускладнити їхнє використання та налагодження.

Низька продуктивність. Використання деяких бібліотек може уповільнити роботу програми, особливо якщо вони не оптимізовані або використовуються неправильно.

Уразливості безпеки. Деякі бібліотеки можуть містити вразливості безпеки, які можуть бути використані зловмисниками для атаки на програму або систему. Важливо ретельно обирати бібліотеки, досліджувати їх перед використанням, перевіряти на відповідність вимогам проєкту і стежити за оновленнями.

У розробці програмного забезпечення для мікроконтролерів поширеними є три підходи до організації повторно використовуваного коду: header-only бібліотеки, звичайні С-бібліотеки та статичні бібліотеки. Кожен з них має власні особливості реалізації, обмеження та сфери застосування. Header-only бібліотека — це бібліотека, в якій як інтерфейс, так і реалізація розміщені в одному заголовному файлі. Реалізація зазвичай оформлюється у вигляді static inline функцій або макросів препроцесора. Такий підхід усуває необхідність у лінкуванні окремих об’єктних файлів, оскільки весь код підставляється безпосередньо на етапі препроцесування. Кожна одиниця трансляції, що включає заголовок, отримує власну копію реалізації. Header-only бібліотеки характеризуються мінімальними вимогами до інтеграції, що робить їх зручними для невеликих модулів, допоміжних функцій та простих абстракцій апаратного рівня. Разом з тим, надмірне використання такого підходу може призводити до збільшення обсягу згенерованого коду та ускладнення супроводу у великих проєктах.

Звичайна С-бібліотека базується на класичному поділі між інтерфейсом і реалізацією. Публічний інтерфейс бібліотеки визначається у заголовному файлі (.h), тоді як реалізація функціоналу міститься у відповідному файлі реалізації (.c). Під час компіляції кожен файл реалізації перетворюється на окремий об’єктний файл, який на етапі лінкування об’єднується з іншими модулями проєкту. Такий підхід забезпечує єдину реалізацію функцій без дублювання коду між одиницями трансляції. Цей метод є базовим та найбільш поширеним у вбудованих системах, оскільки забезпечує чітке розмежування відповідальності, кращу масштабованість та зручність супроводу. Він також дозволяє приховувати внутрішні деталі реалізації за допомогою локальних (статичних) функцій.

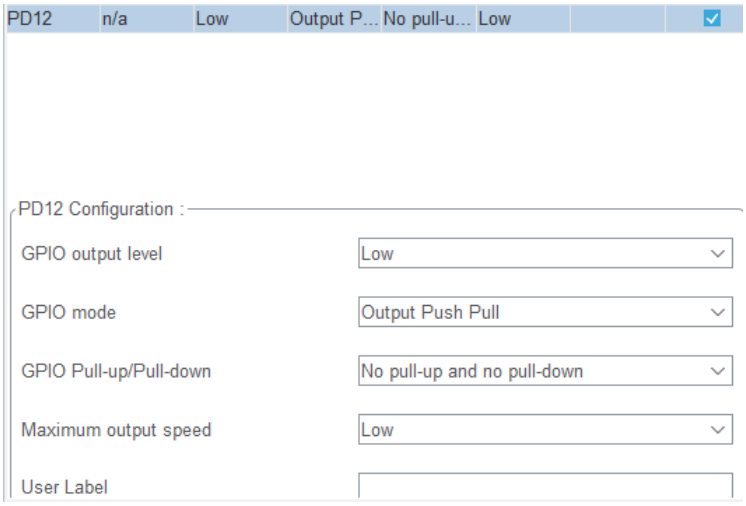
Статична бібліотека являє собою архів об’єктних файлів, об’єднаних у єдиний файл з розширенням .a. Така бібліотека створюється окремо від основного застосунку та підключається на етапі лінкування. Під час побудови виконуваного файлу лінкер вибірково включає до складу прошивки лише ті об’єктні модулі зі статичної бібліотеки, які містять необхідні символи. У результаті код бібліотеки фізично вбудовується у фінальний бінарний файл. Статичні бібліотеки широко застосовуються для багаторазового використання модулів у різних проєктах, розділення процесів розробки та постачання програмних компонентів, а також для приховування реалізації при збереженні відкритого інтерфейсу. Водночас вони вимагають узгодження параметрів компіляції та архітектури цільової платформи між бібліотекою та застосунком.

Хід виконання роботи

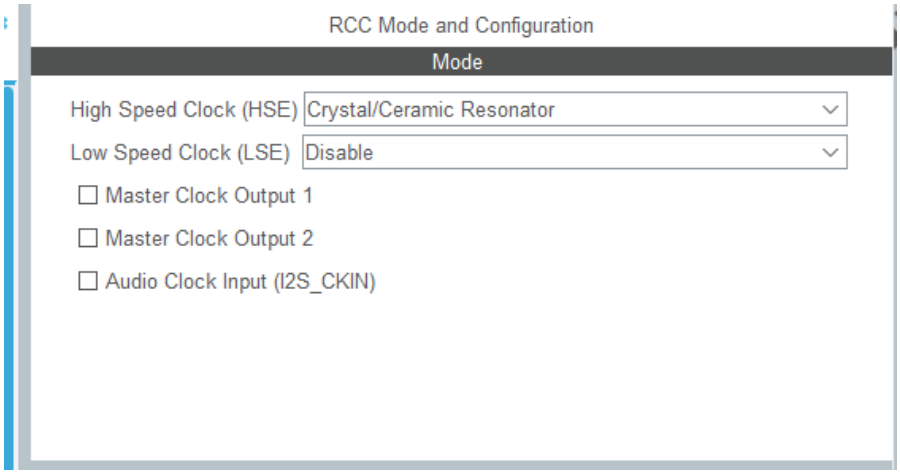
1. Реалізувати відображення стану завантаження причому реалізувати API в окрему Header-only бібліотеку для повторного використання.

1.1 З документації дізнатись данні про піни, що відповідають світлодіодам, червоного, синього, помаранчевого та синього кольорів, що розташовані по колу.

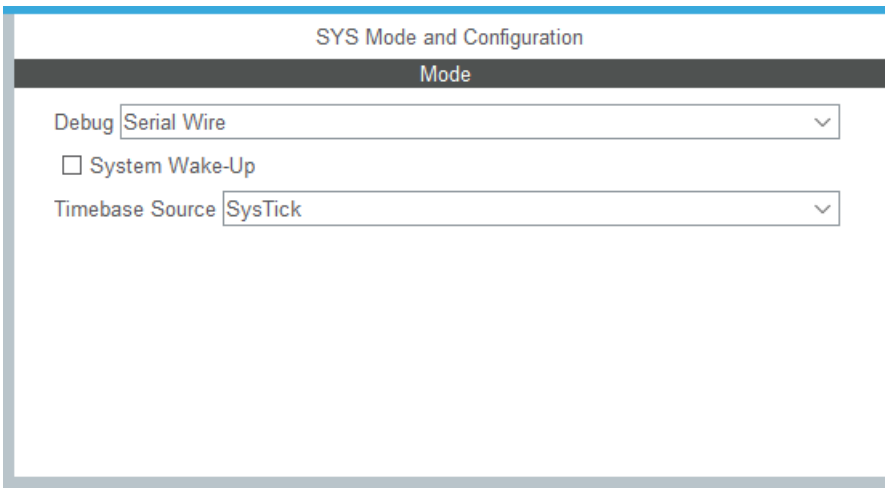
1.2 Налаштувати кожен з 4х пінів наступним чином:



1.3 В RCC налаштувати High Speed Clock ->Cristal/Ceramic Resonator



1.4 В SYS налаштувати Debug -> Serial Wire та згенерувати код натиснувши Ctrl+S.



1.5 В папці Core->Inc створити файл led_loader.h;
1.6 Записати в створений файл код з додатку 1;

1.7 Відкрити Core/Src/main.c та додати include зверху (після стандартних):

```
#include "led_loader.h"
```

1.8 У main() після MX_GPIO_Init(); додати:

```
/* USER CODE BEGIN 2 */
```

```
  LedLoader loader;
  loader_init(&loader, 120);
  /* USER CODE END 2 */
```

1.9 У while(1) додати:

```
loader_update(&loader);
```

1.10 Збілдити код та записати на мікроконтролер і перевірити роботу, критерієм правильної роботи буде послідовне миготіння світлодіодів 4х кольорів по колу. Зробити висновки.

2. Реалізувати відображення стану завантаження причому реалізувати API в окрему С-бібліотека (.c + .h) для повторного використання.

2.1 У Project Explorer стоворити нову папку libs. Правою кнопкою миші -> New → Source Folder**;**

2.3 В libs створи підпапки:

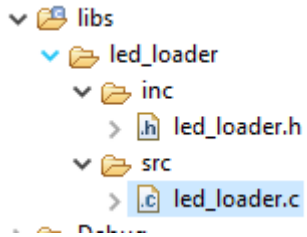
libs/led_loader/inc

libs/led_loader/src;

2.4 Створи файли бібліотеки:

Правою на libs/led_loader/inc → New → File -> led_loader.h

Правою на libs/led_loader/src → New → File -> led_loader.c



2.5 Вставити код у створені файли з додатку 2 відповідно;

2.6 Додати include path (щоб #include "led_loader.h" працював)

Project → Properties → C/C++ General → Paths and Symbols → Вкладка Includes → Вибрати конфігурацію GNU C → Натисни Add… додати шлях (від кореня проєкту): libs/led_loader/inc → Apply and Close

2.7 Замінити код в main.c

```
/* USER CODE BEGIN 2 */
  LedLoader loader;
  LedLoader_Init(&loader, 120); // 120 мс
  /* USER CODE END 2 */
```

та

```
while (1)
  {
    /* USER CODE END WHILE */
        LedLoader_Update(&loader);
    /* USER CODE BEGIN 3 */
  }
```

2.8 Збілдити код та записати на мікроконтролер і перевірити роботу, критерієм правильної роботи буде послідовне миготіння світлодіодів 4х кольорів по колу. Зробити висновки.

3 Реалізувати відображення стану завантаження причому реалізувати API в окрему статичну бібліотеку(.a) для повторного використання.

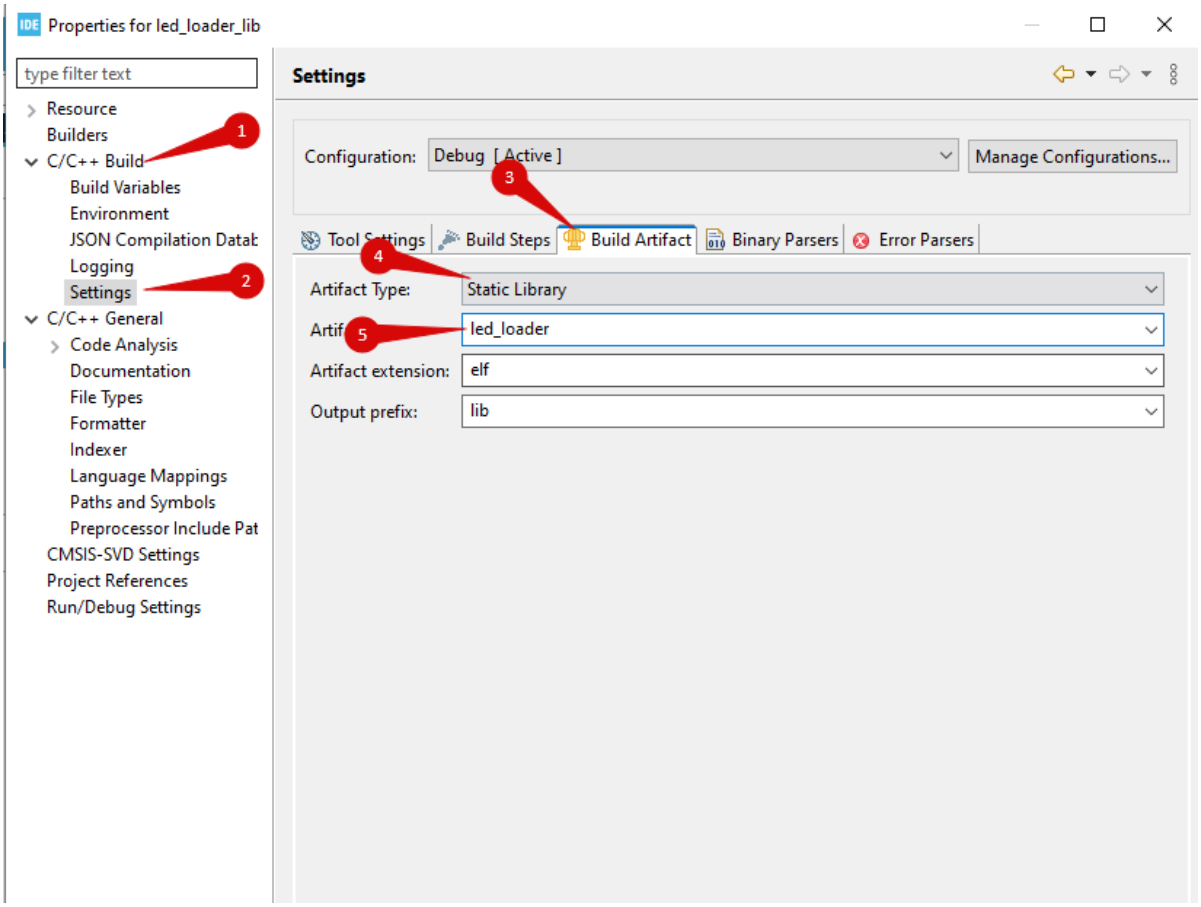
3.1 Створення проєкту бібліотеки в STM32CubeIDE обрати: File → New → STM32 Project Обрати STM32F411VET6

Назва проєкту: led_loader_lib, на етапі налаштувань: тип проєкту: Empty / No CubeMX init, мова: C та Target Binary Type: Static Library;

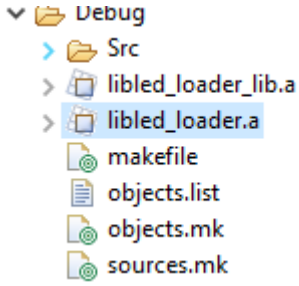
Завершити створення проєкту.

3.2 Відкрити: Project → Properties, перейти: C/C++ Build → Settings, уполі Artifact type обрати: Static Library, у полі Artifact name задати: led_loader

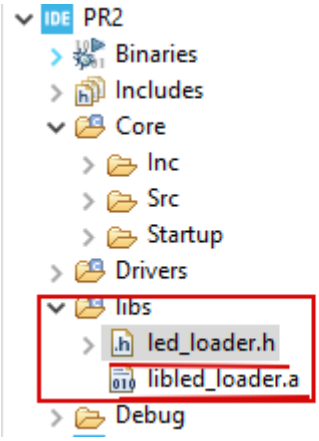
Застосувати зміни (Apply and Close).



3.3 Створити led_loader.h в папці inc та led_loader.c в папці src та вставити код відповідно з додатку 3. В даному коді позбуваємось від залежності HAL бібліотеки. Та збілдити проект. При правильному виконанні має створитись файли Debug.

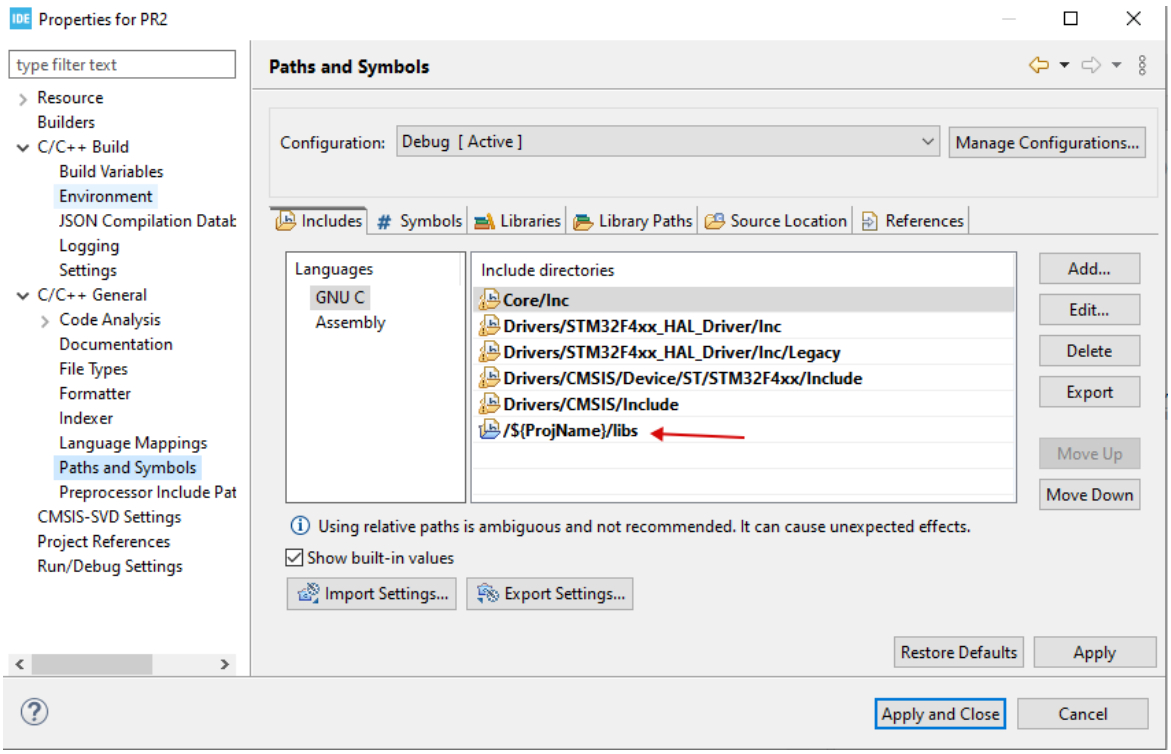


3.4 Очистити в основному проекті файли котрі були створені на попередніх етапах. Створити папку libs та скопіювати файли з статичної бібліотеки.

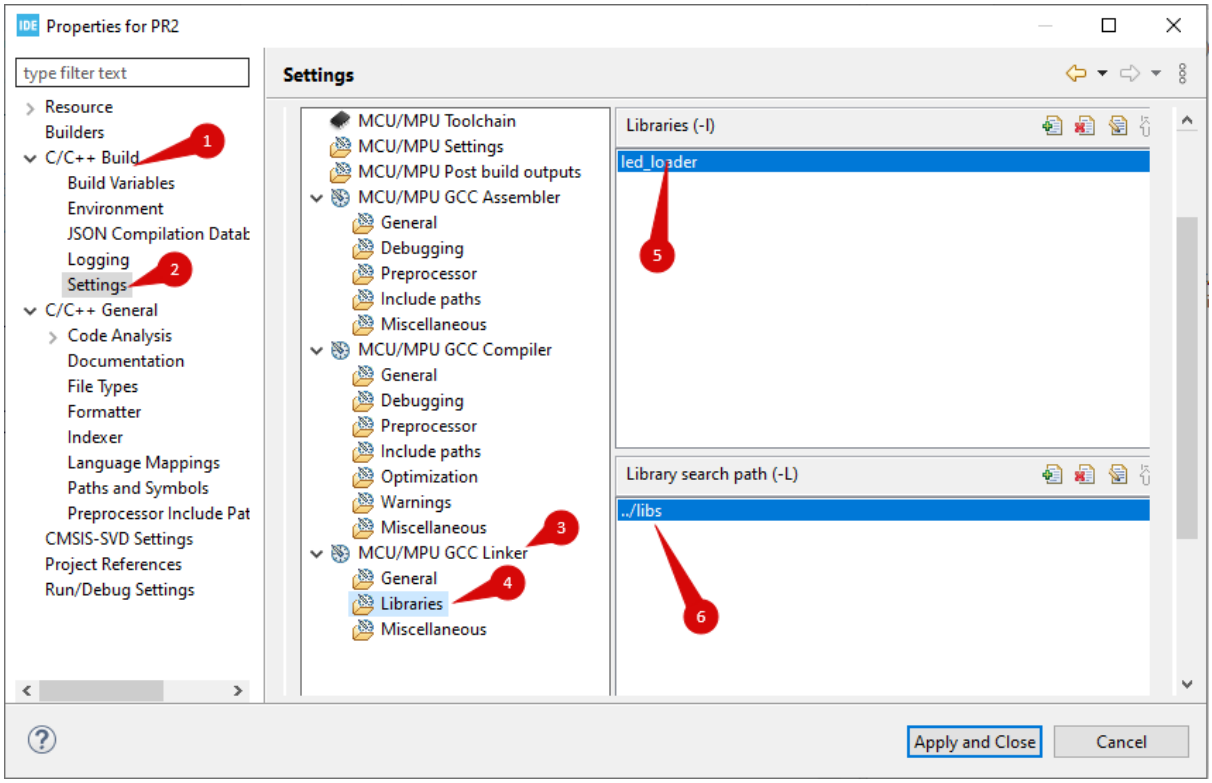


3.5 Додати include path (щоб #include "led_loader.h" працював)

Project → Properties → C/C++ General → Paths and Symbols → Вкладка Includes → Вибрати конфігурацію GNU C → Натисни Add… додати шлях (від кореня проєкту): libs → Apply and Close



Після чого додати бібліотеку лінкеру



3.6 У файл проекту main.c додати

```
#include "led_loader.h"
```

Та наступний код

```
static void LedWrite(uint8_t index, uint8_t on)
{
  uint16_t pin;
  switch (index) {
    case 0: pin = GPIO_PIN_12; break;
    case 1: pin = GPIO_PIN_13; break;
    case 2: pin = GPIO_PIN_14; break;
    default: pin = GPIO_PIN_15; break;
  }

  HAL_GPIO_WritePin(GPIOD, pin,
                    on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static uint32_t Tick(void)
{
  return HAL_GetTick();
}


int main(void)
{
        HAL_Init();
         SystemClock_Config();
         MX_GPIO_Init();

         LedLoader loader;
         LedLoader_Init(&loader, 120, LedWrite, Tick);

         while (1) {
           LedLoader_Update(&loader);
         }
  /* USER CODE END 3 */
```

}
3.7 Збілдити код та записати на мікроконтролер і перевірити роботу, критерієм правильної роботи буде послідовне миготіння світлодіодів 4х кольорів по колу. Зробити висновки.

Зміст звіту

1. Тема та мета роботи;

2. Скріншот файлу .ioc з конфігурацією усіх задіяних пінів контролера;

3. Скріншот ієрархії папок та код(main.c, led_loader.h) по пункту 1 ходу виконання роботу;

4. Скріншот ієрархії проекту, а саме папки libs та код(main.c, led_loader.h, led_loader.c) по пункту 2 ходу виконання роботу;

5. Скріншот ієрархії проекту, а саме папки libs та Static library, по пункту 3 ходу виконання роботу;

6. Висновки по кожному пункту.

Контрольні запитання

1. Що таке статична бібліотека (.a) і на якому етапі відбувається її підключення до проєкту?

2. У чому різниця між статичною бібліотекою та звичайним модулем .c/.h?

3. Яке призначення ключа -L у GCC?

4. Що станеться, якщо бібліотека скомпільована для іншого MCU?

```c
#pragma once
#include "main.h"
#include <stdint.h>


typedef struct {
  uint32_t period_ms;  // період кроку
  uint32_t last_ms;    // час останнього кроку
  uint8_t  idx;        // 0..3
} LedLoader;

#ifndef LOADER_LED1_PORT
#define LOADER_LED1_PORT GPIOD
#define LOADER_LED1_PIN  GPIO_PIN_12

#define LOADER_LED2_PORT GPIOD
#define LOADER_LED2_PIN  GPIO_PIN_13

#define LOADER_LED3_PORT GPIOD
#define LOADER_LED3_PIN  GPIO_PIN_14

#define LOADER_LED4_PORT GPIOD
#define LOADER_LED4_PIN  GPIO_PIN_15
#endif

static inline void loader_all_off(void)
{
  HAL_GPIO_WritePin(LOADER_LED1_PORT, LOADER_LED1_PIN, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LOADER_LED2_PORT, LOADER_LED2_PIN, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LOADER_LED3_PORT, LOADER_LED3_PIN, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LOADER_LED4_PORT, LOADER_LED4_PIN, GPIO_PIN_RESET);
}

static inline void loader_one_on(uint8_t i)
{
  loader_all_off();

  switch (i & 3u) {
    case 0: HAL_GPIO_WritePin(LOADER_LED1_PORT, LOADER_LED1_PIN, GPIO_PIN_SET); break;
    case 1: HAL_GPIO_WritePin(LOADER_LED2_PORT, LOADER_LED2_PIN, GPIO_PIN_SET); break;
    case 2: HAL_GPIO_WritePin(LOADER_LED3_PORT, LOADER_LED3_PIN, GPIO_PIN_SET); break;
    default:HAL_GPIO_WritePin(LOADER_LED4_PORT, LOADER_LED4_PIN, GPIO_PIN_SET); break;
  }
}

static inline void loader_init(LedLoader* h, uint32_t period_ms)
{
  if (!h) return;
  h->period_ms = (period_ms ? period_ms : 100u);
  h->last_ms   = HAL_GetTick();
  h->idx       = 0u;
  loader_one_on(h->idx);
}

static inline void loader_update(LedLoader* h)
{
  if (!h) return;
  uint32_t now = HAL_GetTick();
  if ((now - h->last_ms) >= h->period_ms) {
    h->last_ms = now;
    h->idx = (uint8_t)((h->idx + 1u) & 3u);
    loader_one_on(h->idx);
```

```
  }
}
```

Додаток 2

led_loader.h

```
#pragma once
#include "main.h"
#include <stdint.h>

typedef struct {
  uint32_t period_ms;
  uint32_t last_ms;
  uint8_t  idx;
} LedLoader;

void LedLoader_Init(LedLoader* h, uint32_t period_ms);
void LedLoader_Update(LedLoader* h);
void LedLoader_Stop(void);
```

led_loader.c

```
#include "led_loader.h"


#define LED_MASK (GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15)

static void AllOff(void)
{
  HAL_GPIO_WritePin(GPIOD, LED_MASK, GPIO_PIN_RESET);
}

static void OneOn(uint8_t i)
{
  AllOff();
  switch (i & 3u) {
    case 0: HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET); break;
    case 1: HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET); break;
    case 2: HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); break;
    default:HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET); break;
  }
}

void LedLoader_Init(LedLoader* h, uint32_t period_ms)
{
  if (!h) return;
  h->period_ms = (period_ms ? period_ms : 100u);
  h->last_ms   = HAL_GetTick();
  h->idx       = 0u;
  OneOn(h->idx);
}

void LedLoader_Update(LedLoader* h)
{
  if (!h) return;
  uint32_t now = HAL_GetTick();
  if ((now - h->last_ms) >= h->period_ms) {
    h->last_ms = now;
    h->idx = (uint8_t)((h->idx + 1u) & 3u);
    OneOn(h->idx);
```

```
  }
}

void LedLoader_Stop(void)
{
  AllOff();
}
```

Додаток 3

led_loader.h

```
#pragma once
#include <stdint.h>

typedef void (*LedWriteFn)(uint8_t index, uint8_t on);
typedef uint32_t (*TickFn)(void);

typedef struct {
  uint32_t period_ms;
  uint32_t last_ms;
  uint8_t  idx;
  LedWriteFn write;
  TickFn tick;
} LedLoader;

void LedLoader_Init(LedLoader* h, uint32_t period_ms,
                    LedWriteFn write,
                    TickFn tick);

void LedLoader_Update(LedLoader* h);
void LedLoader_Stop(LedLoader* h);
```

led_loader.c

```
#include "led_loader.h"

static void all_off(LedLoader* h)
{
  for (uint8_t i = 0; i < 4; i++)
    h->write(i, 0);
}

void LedLoader_Init(LedLoader* h,
                    uint32_t period_ms,
                    LedWriteFn write,
                    TickFn tick)
{
  if (!h || !write || !tick) return;

  h->period_ms = period_ms ? period_ms : 100;
  h->write = write;
  h->tick = tick;
  h->last_ms = h->tick();
  h->idx = 0;

  all_off(h);
  h->write(0, 1);
}

void LedLoader_Update(LedLoader* h)
{
  uint32_t now = h->tick();
```

```
  if ((now - h->last_ms) >= h->period_ms) {
    h->last_ms = now;
    h->idx = (h->idx + 1) & 3;

    all_off(h);
    h->write(h->idx, 1);
  }
}

void LedLoader_Stop(LedLoader* h)
{
  all_off(h);
}
```