****

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №1  
Проектування вбудованих систем

«Створення програмного забезпечення для Вбудованих Систем за допомогою симулятора wokwi та середовища VS code»

Варіант 17

Виконав Перевірив:

студент групи ІА–13: Гордієнко Н.Ю.

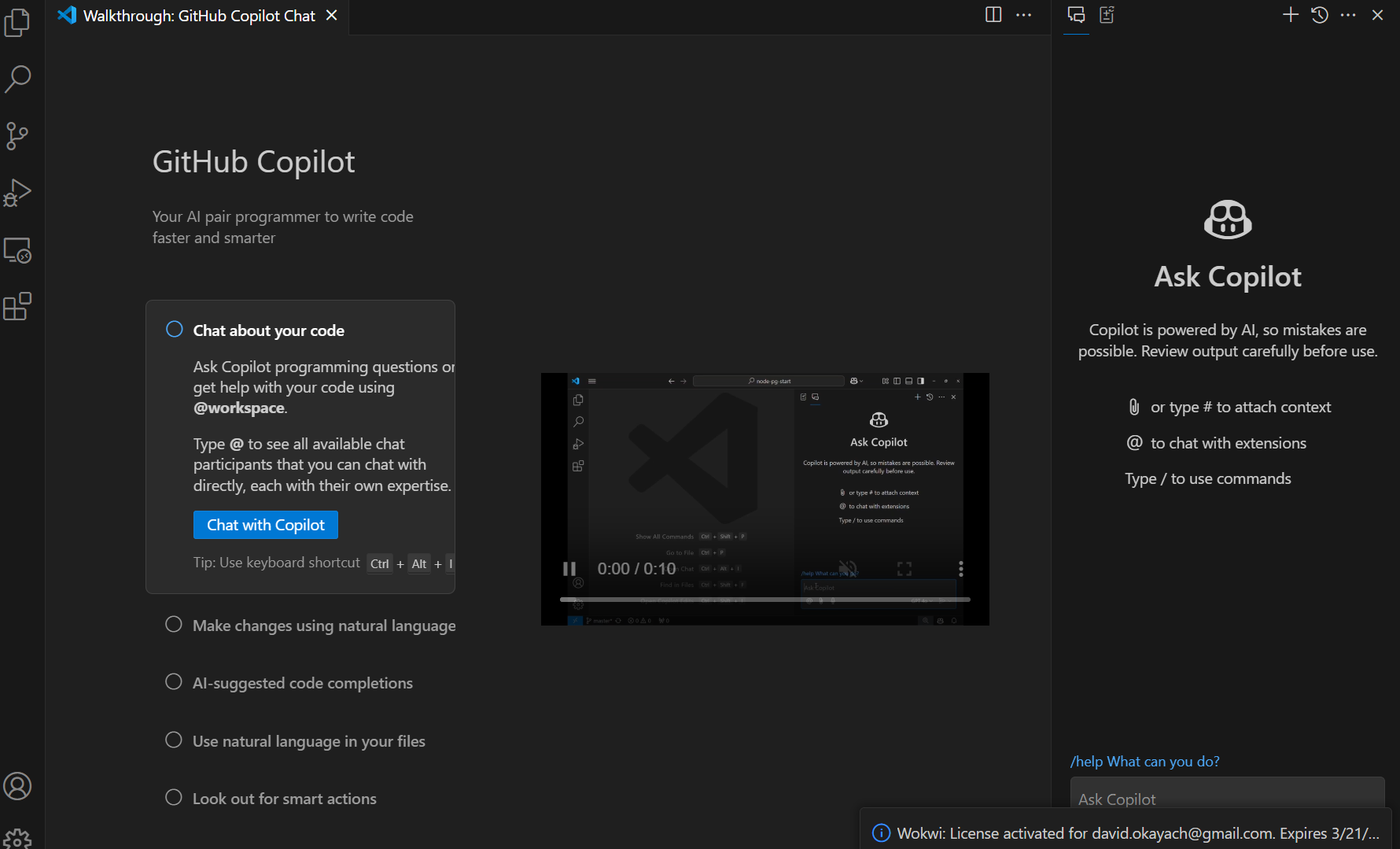
Окаянченко Д.О.

Київ 2025

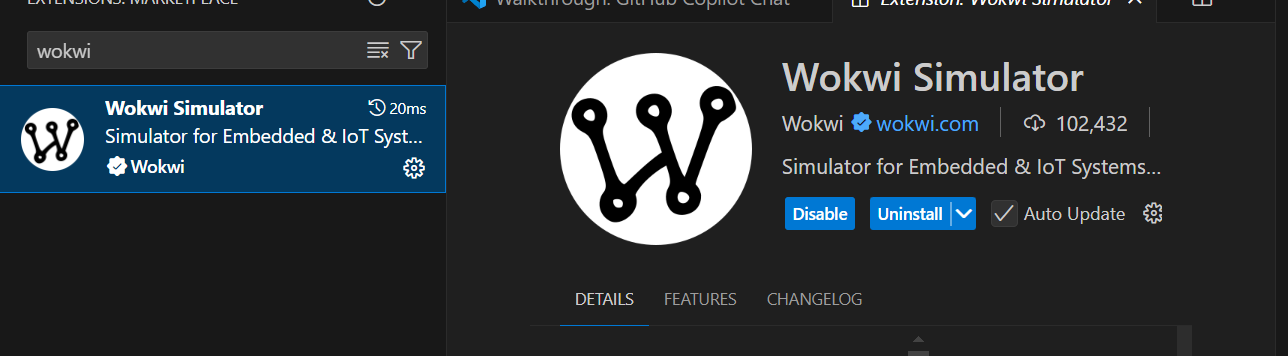
**Мета роботи:** Навчитись працювати з середовищем для розробки та проектування вбудованих систем на основі симулятора вбудованих систем Wokwi.

## Хід роботи

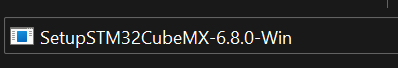
1. Встановимо VS code:

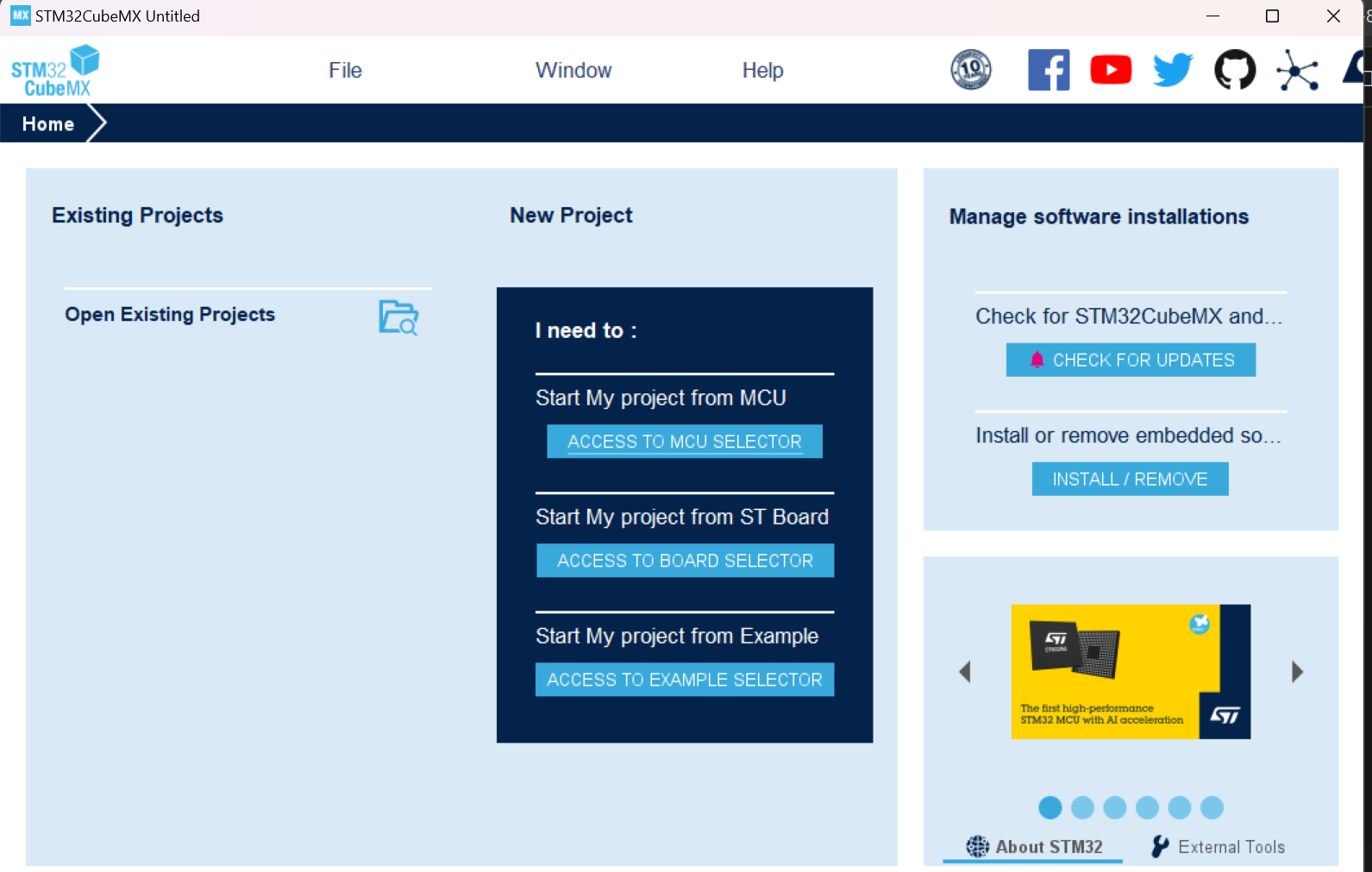


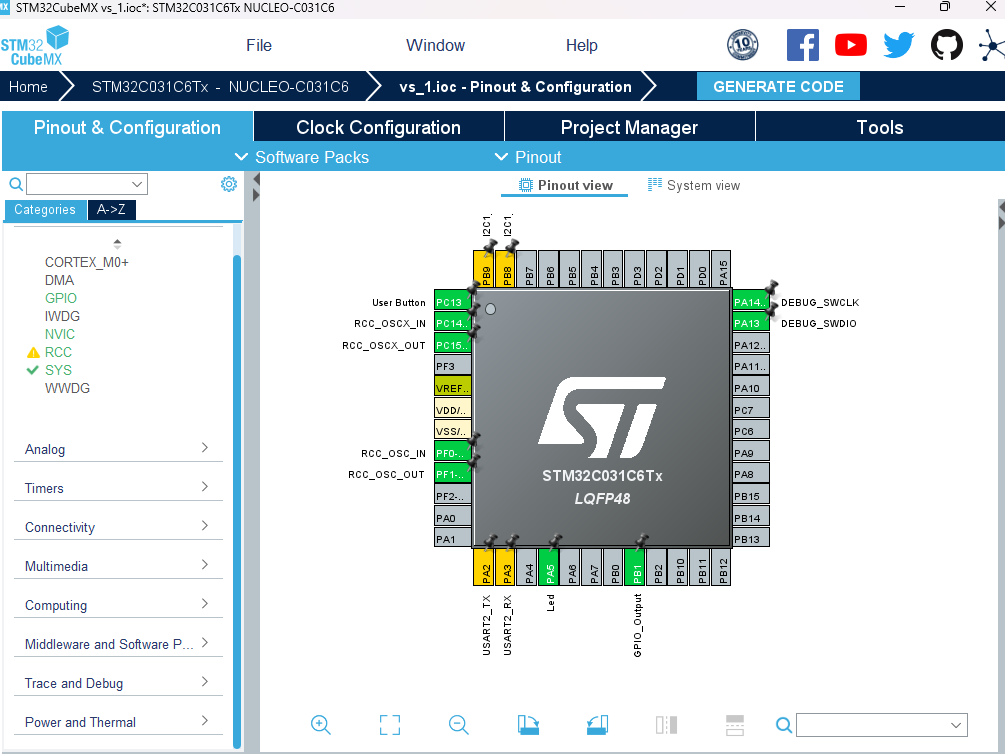
1. Встановимо Wokwi в VS code:



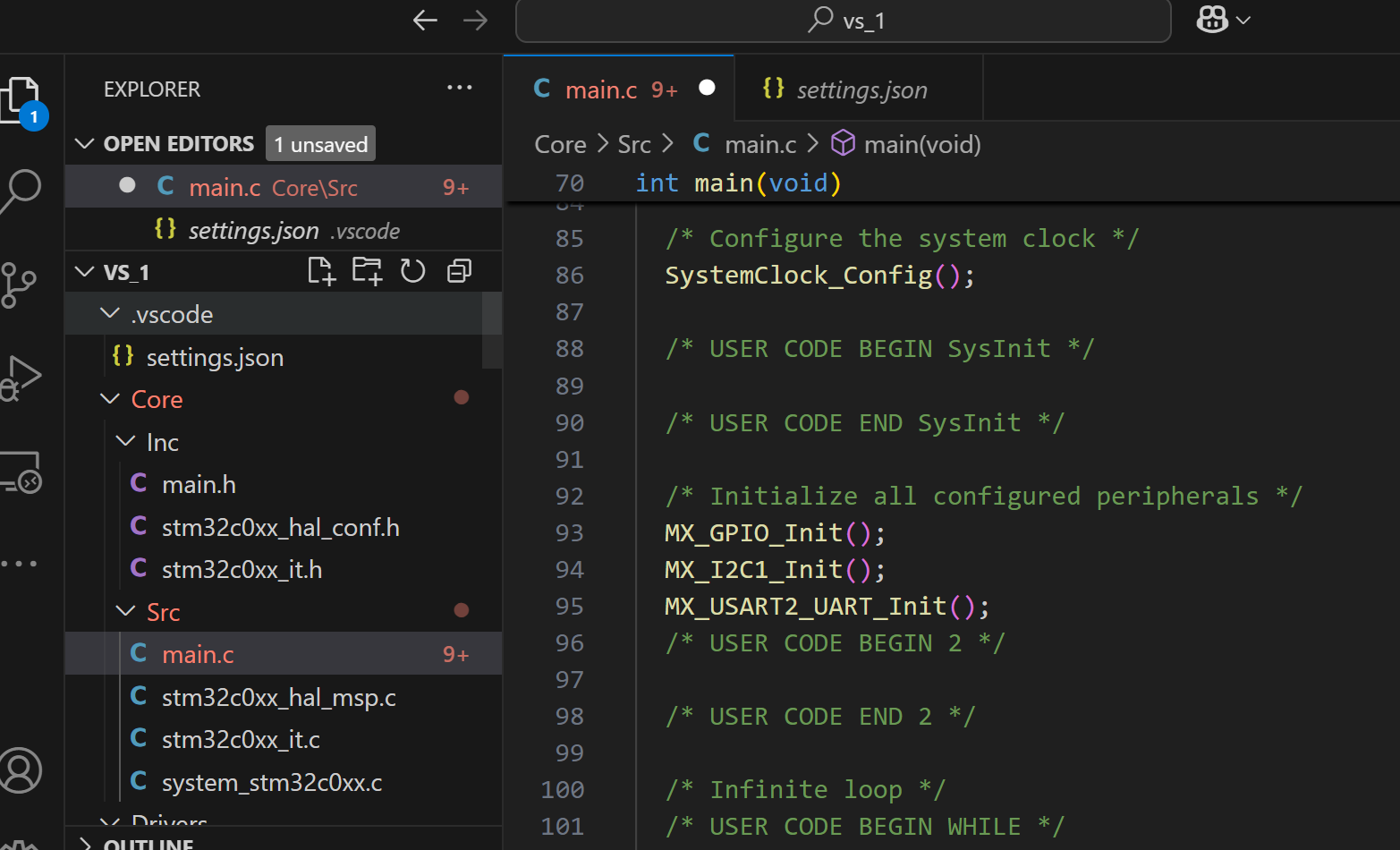
1. Встановимо Stm32cubemx та створити проєкт на основі STM32C031C6:



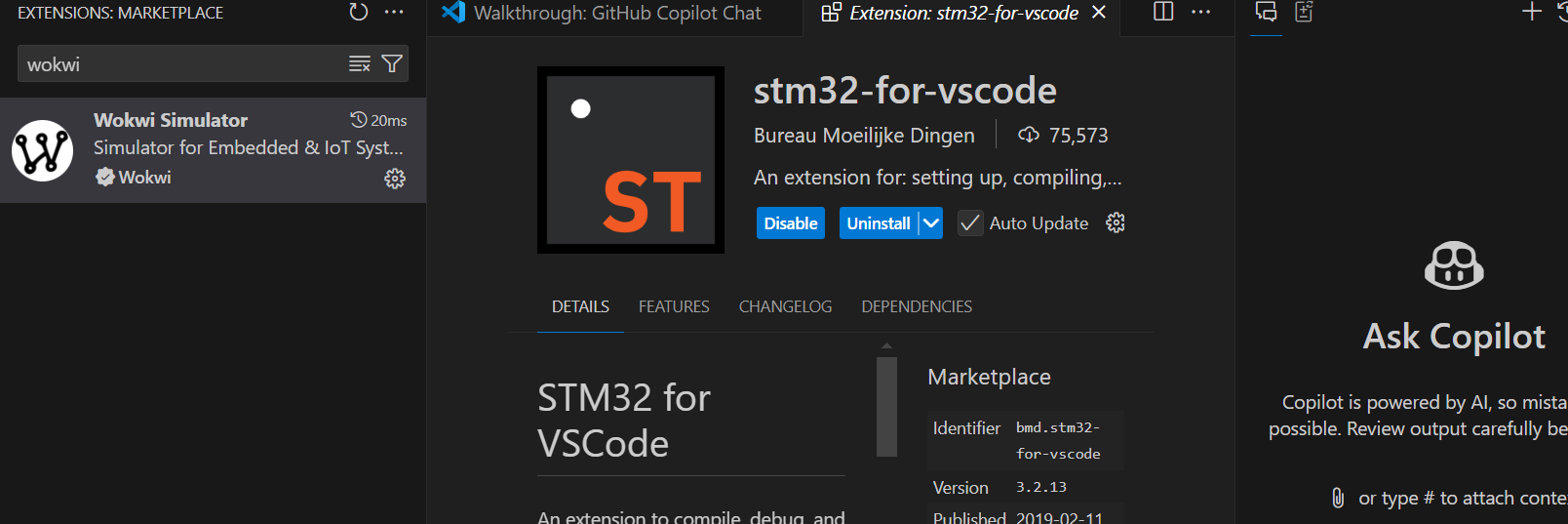




1. Відкриємо проєкт у VS code:



1. Встановимо наступний додаток, щоб легко компілювати проєкт:



1. Створимо та запустимо програму, яка блимає світлодіодом:

**Код:**

/\* USER CODE BEGIN Header \*/

/\*\*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @file           : main.c

  \* @brief          : Main program body

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @attention

  \*

  \* 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 \*/

/\* Includes ------------------------------------------------------------------\*/

#include "stm32c0xx\_hal.h"

#include "stm32c0xx\_hal\_gpio.h"

#include "stm32c0xx\_hal\_rcc.h"

/\* Function prototypes \*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

void Custom\_Error\_Handler(void);

/\*\*

  \* @brief  The application entry point.

  \* @retval int

  \*/

int main(void)

{

  /\* Initialize the HAL Library \*/

  HAL\_Init();

  /\* Configure the system clock \*/

  SystemClock\_Config();

  /\* Initialize all configured peripherals \*/

  MX\_GPIO\_Init();

  /\* Infinite loop \*/

  while (1)

  {

    /\* Toggle LED connected to GPIOB Pin 0 \*/

    HAL\_GPIO\_TogglePin(GPIOB, GPIO\_PIN\_1);

    /\* Delay for 500ms \*/

    HAL\_Delay(500);

  }

}

/\*\*

  \* @brief System Clock Configuration

  \* @retval None

  \*/

void SystemClock\_Config(void)

{

  RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

  RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

  /\* Configure Flash Latency \*/

  \_\_HAL\_FLASH\_SET\_LATENCY(FLASH\_LATENCY\_1);

  /\* Initializes the RCC Oscillators \*/

  RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

  RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

  RCC\_OscInitStruct.HSIDiv = RCC\_HSI\_DIV1;

  RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

  if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

  {

    Custom\_Error\_Handler();

  }

  /\* 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\_HSI;

  RCC\_ClkInitStruct.SYSCLKDivider = RCC\_SYSCLK\_DIV1;

  RCC\_ClkInitStruct.AHBCLKDivider = RCC\_HCLK\_DIV1;

  RCC\_ClkInitStruct.APB1CLKDivider = RCC\_APB1\_DIV1;

  if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_1) != HAL\_OK)

  {

    Custom\_Error\_Handler();

  }

}

/\*\*

  \* @brief GPIO Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_GPIO\_Init(void)

{

  GPIO\_InitTypeDef GPIO\_InitStruct = {0};

  /\* Enable clock for GPIOB \*/

  \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

  /\* Configure GPIO pin Output Level \*/

  HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_RESET);

  /\* Configure GPIO pin : PB1 \*/

  GPIO\_InitStruct.Pin = 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(GPIOB, &GPIO\_InitStruct);

}

/\*\*

  \* @brief This function is executed in case of error occurrence.

  \* @retval None

  \*/

void Custom\_Error\_Handler(void)

{

  \_\_disable\_irq();

  while (1)

  {

    /\* Stay in loop \*/

  }

}

#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)

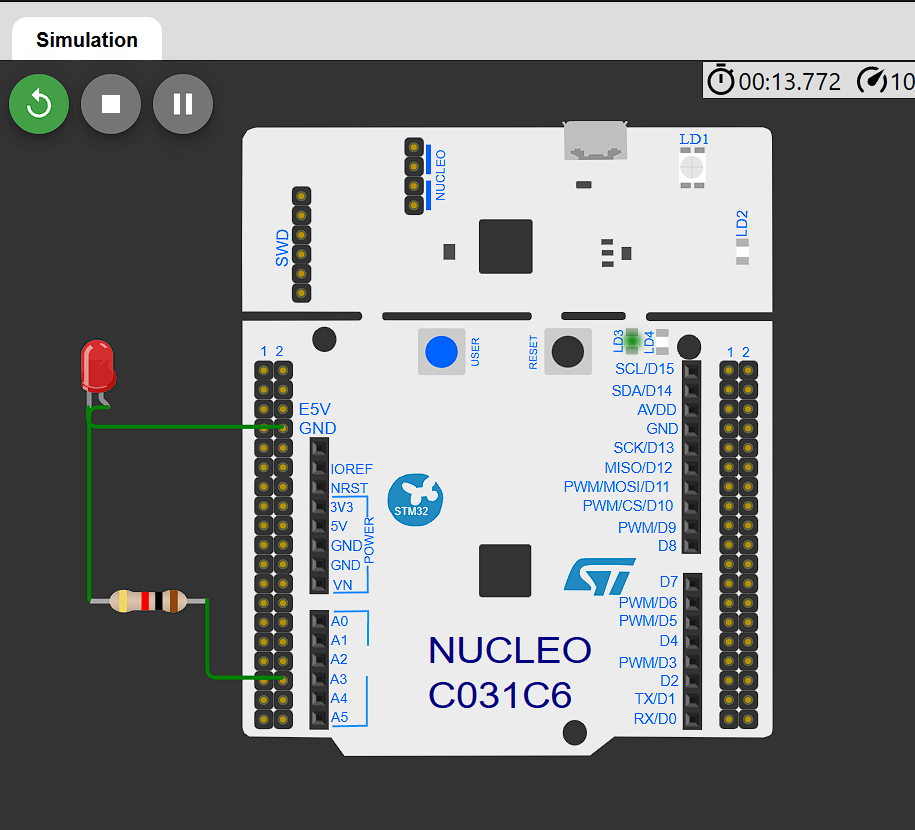
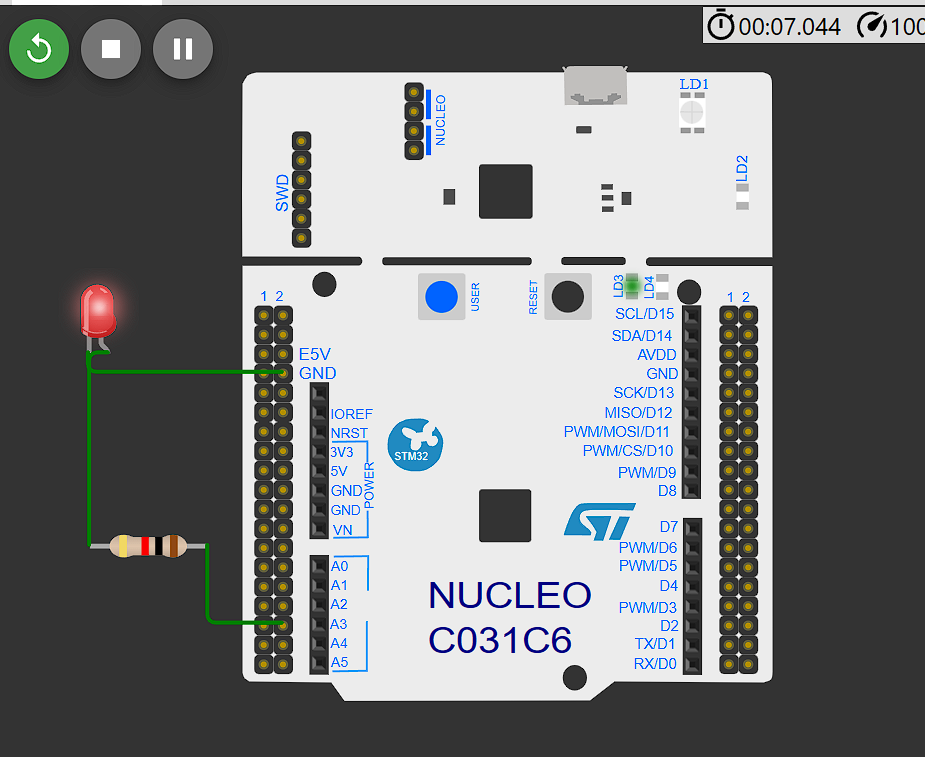
{

  /\* User can add their own implementation to report the file name and line number \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

**Результат на платі:**

****

Підключили резистор до піна PB0, резистор до LED, другий вихід якого провели до GND.

**Посилання на проєкт на wokwi:** <https://wokwi.com/projects/423335388190436353>.

1. Ознайомимося з <https://docs.wokwi.com/parts/wokwi-microsd-card>, проаналізуємо далі модуль Protocol Decoder:

Протокольний декодер — це спеціалізований інструмент, що дозволяє аналізувати та інтерпретувати цифрові сигнали відповідно до певних комунікаційних протоколів. Він перетворює низькорівневі електричні сигнали на зрозумілі дані, що значно полегшує процес діагностики та налагодження електронних систем.

У сфері розробки та тестування електроніки протокольний декодер є незамінним помічником для інженерів. Він допомагає перевіряти коректність передачі даних між різними компонентами, такими як мікроконтролери, датчики та периферійні пристрої. Завдяки цьому можна виявити та усунути можливі несправності або неточності в роботі системи на ранніх етапах розробки.

В освітньому процесі протокольний декодер служить ефективним засобом для демонстрації роботи різних протоколів передачі даних. Студенти та викладачі можуть використовувати його для вивчення принципів функціонування комунікаційних протоколів, що сприяє глибшому розумінню роботи сучасних електронних систем.

У галузі ремонту та обслуговування електронного обладнання протокольний декодер дозволяє технічним спеціалістам швидко виявляти та усувати несправності в комунікаційних лініях. Аналізуючи сигнали, він допомагає визначити причину проблеми та забезпечити оперативне відновлення працездатності системи.

Протокольний декодер ефективно працює в поєднанні з такими інструментами, як логічні аналізатори та осцилографи. Логічні аналізатори забезпечують захоплення цифрових сигналів, які потім декодуються для детального аналізу. Осцилографи, в свою чергу, візуалізують сигнали в реальному часі, а протокольний декодер інтерпретує ці сигнали відповідно до обраного протоколу. Крім того, у поєднанні з мікроконтролерами та периферійними модулями декодер допомагає перевіряти коректність обміну даними між компонентами системи, такими як датчики, дисплеї або комунікаційні модулі.

Загалом, використання протокольного декодера спрощує процес розробки, тестування та обслуговування електронних систем, забезпечуючи точний аналіз та інтерпретацію цифрових комунікацій.

**Висновок:** У результаті виконання лабораторної роботи було налаштовано середовище розробки, створено проєкт для STM32C031C6 та реалізовано програму керування світлодіодом. Досліджено можливості використання модуля microSD та протокольного декодера. Отримані навички допоможуть у подальшій розробці та аналізі вбудованих систем.