Міністерство освіти та науки України

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

ЛАБОРАТОРНА РОБОТА №4

з курсу «Операційні системи реального часу»

«Формування імпульсного сигналу мікроконтролера з використанням

операційної системи RTOS»

Виконали:

студенти групи ІТ-81

Тищенко І. В.

Юхимчук А. В.

Ачілов А. В.

Перевірив:

Катін П. Ю.

доцент кафедри АУТС

Київ 2021

# Мета лабораторної роботи

Дослідити управління затримками часу у RTOS для формування імпульсного сигналу довільної форми з використанням віртуального таймеру.

Побудувати і дослідити програму для формування імпульсного сигналу з використанням RTOS.

Протестувати механізми налагодження і управління потоками з використанням середовища MDK Keil на основі ядра CMSIS.

**Завдання**

1. Створити і побудувати проект відповідно до описаного далі завдання на базі MDK Keil на основі RTOS.

2. Написати і налагодити програму, що використовує потоки для виконання у ОС RTOS з метою формування сигналу і з використанням віртуального таймеру.

3. Протестувати програму вбудованими засобами MDK Keil.

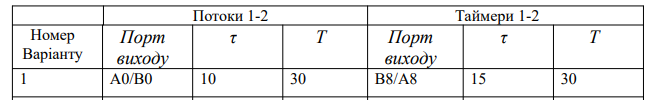
4. Протестувати програму з використанням плати налагодження STM32F103C8.

5. Задокументувати отриманий сигнал і зробити висновки.

**Хід роботи**

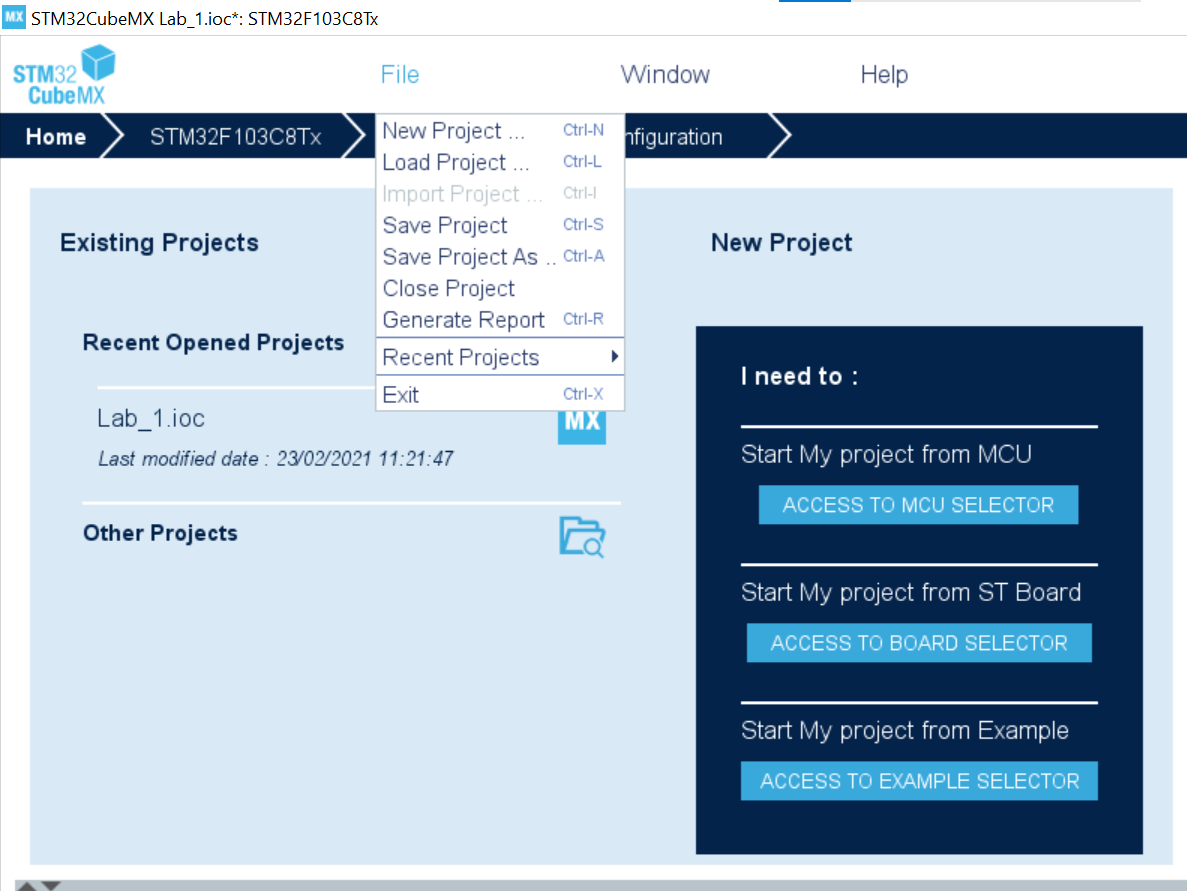
1. У даній ЛР потрібно створити 2 віртуальних таймера і два окремих потоки.
2. Один потік має вищій пріоритет відносно іншого.
3. Всі таймери і потоки формують сигнал на виходах, параметри якого надані у таблиці нижче.
4. Реалізувати програму у 2 варіантах за вибором, користуючись бібліотеками
5. CMSIS для драйверів, або структур CMSIS.

**Варіант 1:**

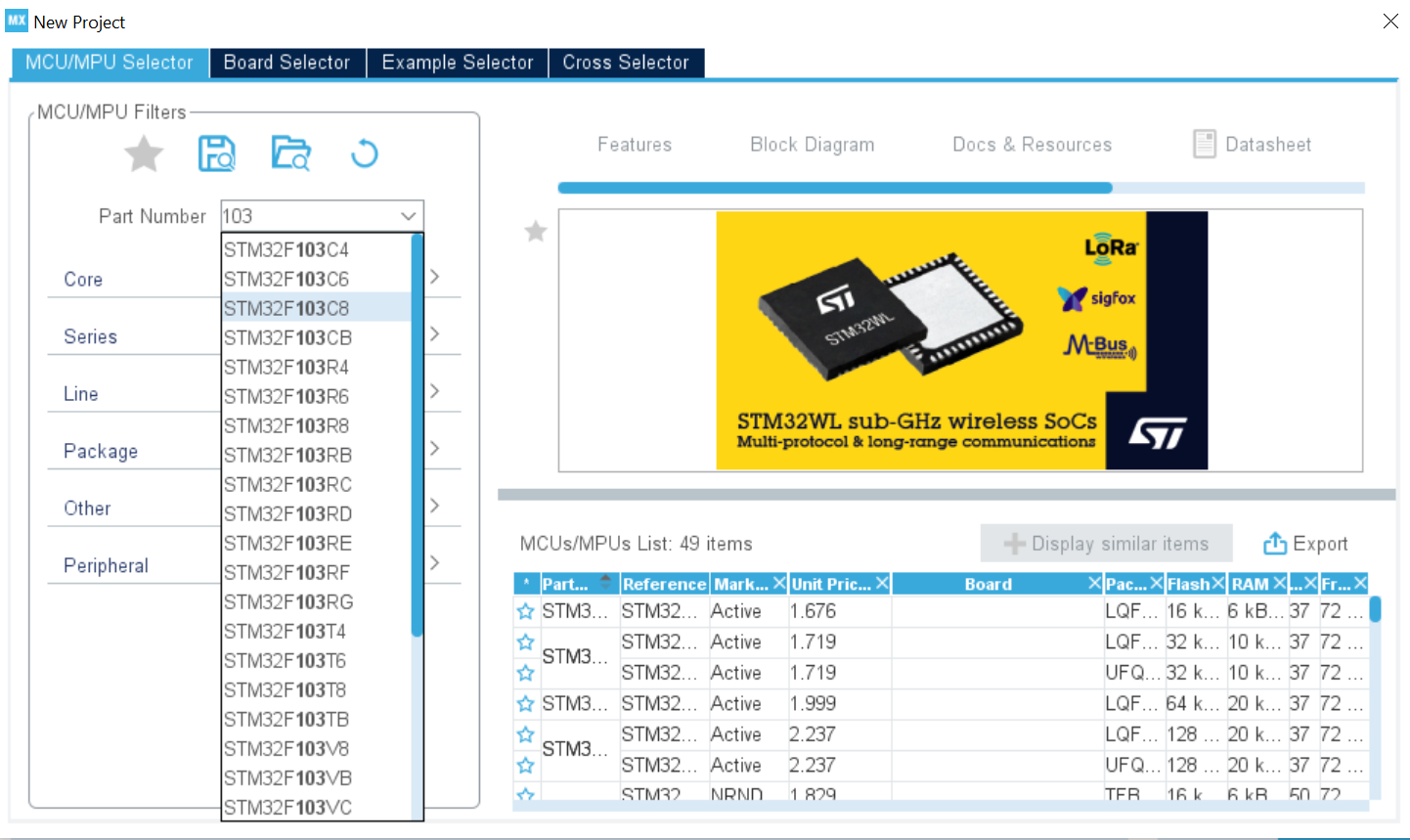


**Виконання роботи**

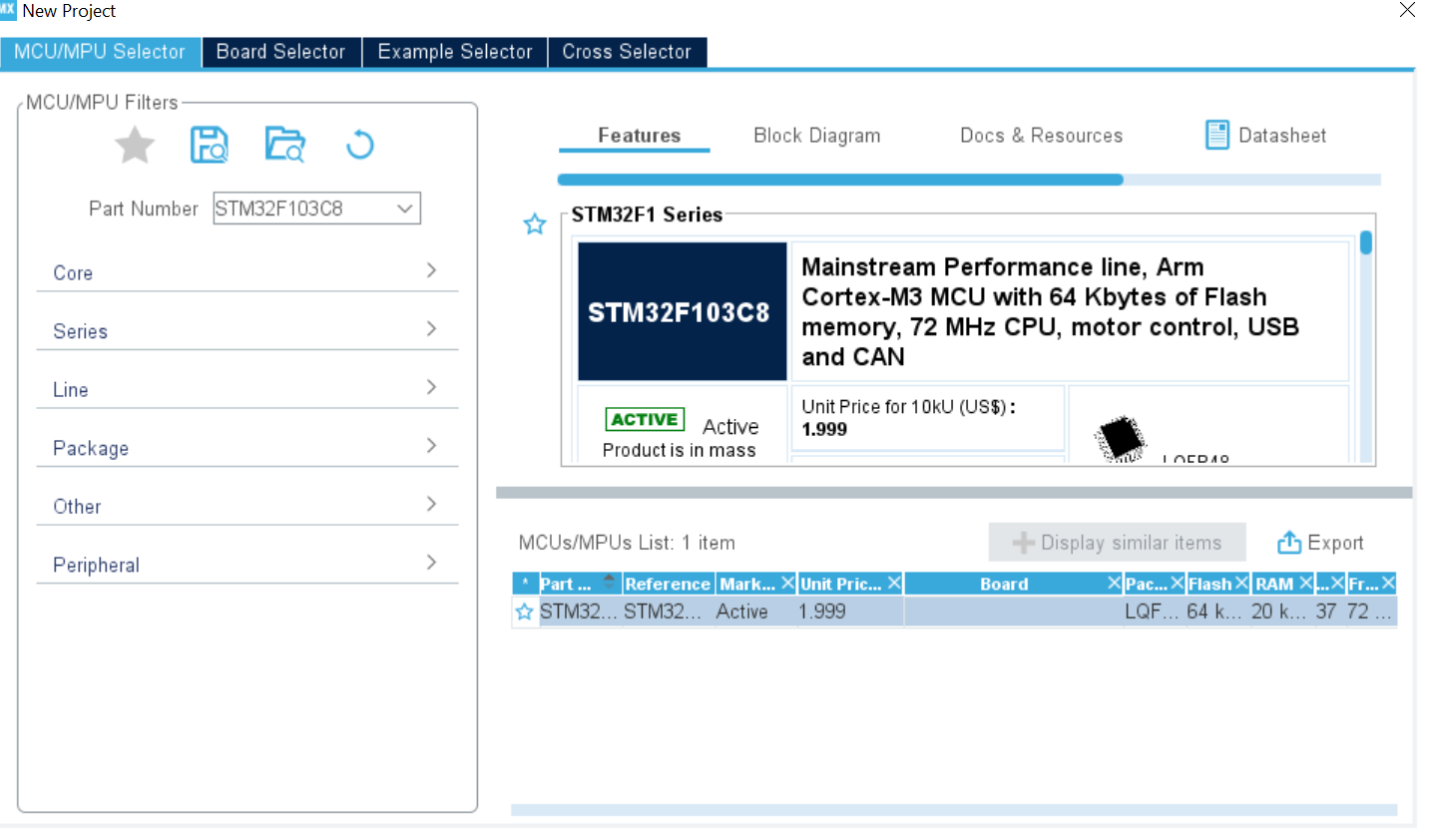
1.Використовуємо середовище CUBEMX і створюємо новий проект



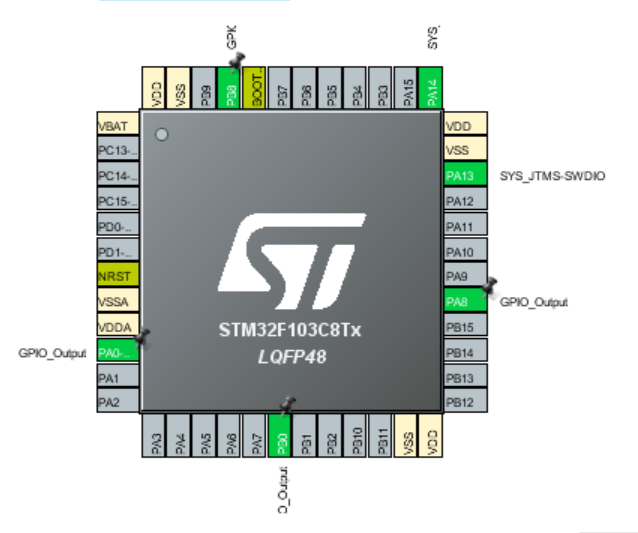
2.Обираємо тип SMTP103C8 мікропроцесора



3.Зберігаємо та перходимо до наступного меню

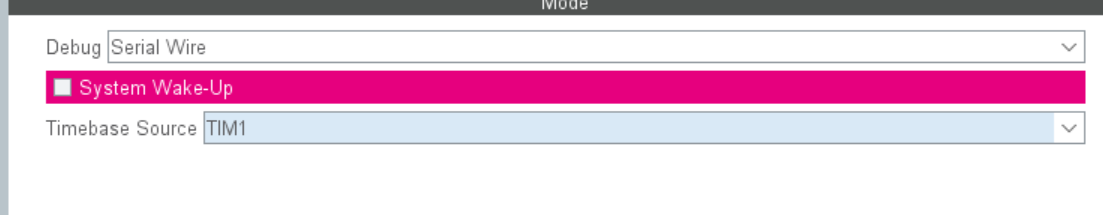


4.Бачимо зображену схему мікропроцесора. По варіанту завдання обираємо пін A0 на вивід, а B0, B1 – на вхід. Пін B6 налаштовуємо на таймер TIM4.

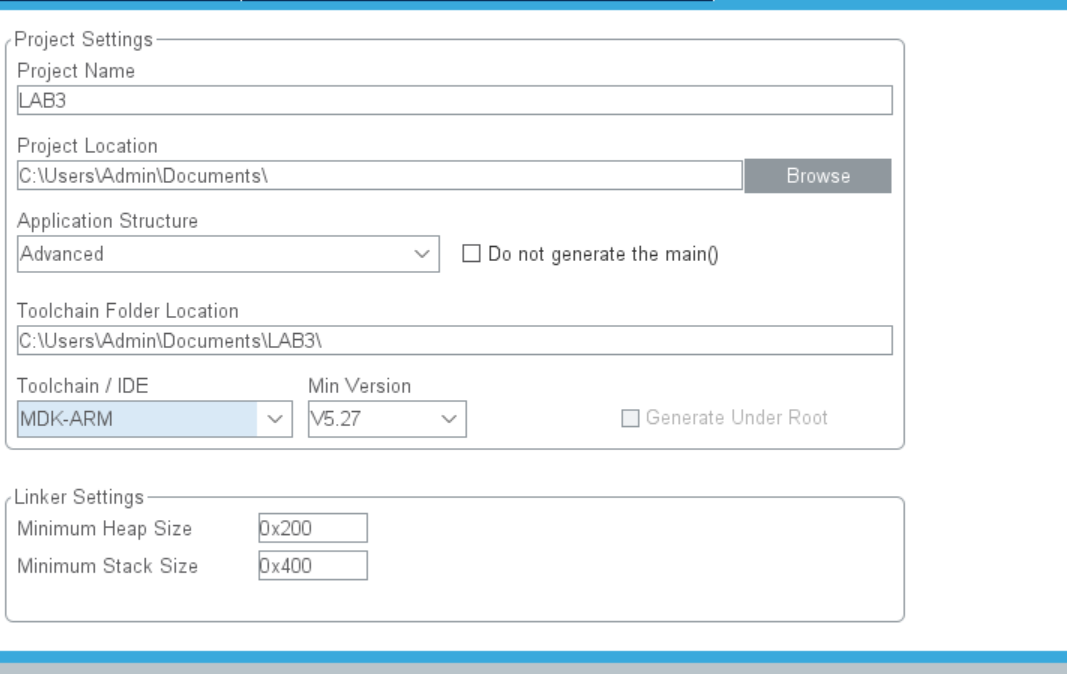


|  |
| --- |
| 5. Виберемо GPIO піни: |

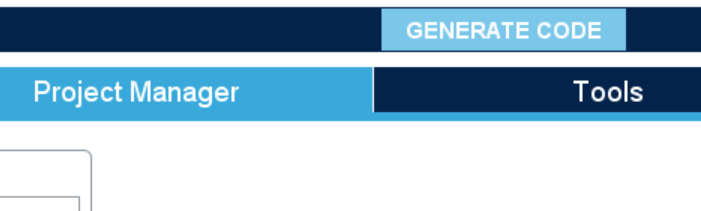
6. Оберемо тип дебагу Serial Ware. TimeBase Source лишаємо за замовчуванням.

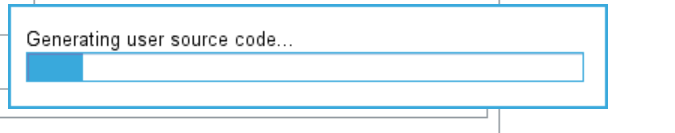


7. Обираємо IDE, розміщення програми та її назву.



8. Натискаємо згенерувати код



9. Очікуємо певний момент часу поки код генерується і відкриваємо IDE із згенерованим кодом.

Структура згенерованого проекту:

10. Додамо новий код до класу main для виконання завдання лабораторної роботи. Отримали наступний вихідний код, який відповідає за формування імпульсного сигналу з заданими хаарктеристиками за допомогою віртуального таймера:

/\* USER CODE BEGIN Header \*/

/\*\*

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

\* @file : main.c

\* @brief : Main program body

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

\* @attention

\*

\* <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.

\* All rights reserved.</center></h2>

\*

\* This software component is licensed by ST under Ultimate Liberty license

\* SLA0044, the "License"; You may not use this file except in compliance with

\* the License. You may obtain a copy of the License at:

\* www.st.com/SLA0044

\*

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

\*/

/\* USER CODE END Header \*/

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

#include "main.h"

#include "cmsis\_os.h"

/\* Private variables ---------------------------------------------------------\*/

TIM\_HandleTypeDef htim2;

volatile uint32\_t ticks\_delay;

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM2\_Init(void);

/\* 2 counters for timers\*/

volatile uint16\_t counter1 = 0;

volatile uint16\_t counter2 = 0;

/\* 2 handlers for threads\*/

osThreadId\_t ThreadId1;

osThreadId\_t ThreadId2;

/\* 2 handlers for timers\*/

osTimerId\_t TimerId1;

osTimerId\_t TimerId2;

/\* virtual timer 1 and 2 callback \*/

void timerCallback(void \*argument);

/\* thread function \*/

void pwmThreadFunc(void \*param);

void \_delay\_ms(uint32\_t ms);

void initGPIO(void);

void PWM(uint16\_t dutyCycle, uint16\_t period, GPIO\_TypeDef\* port, uint16\_t pin);

const osThreadAttr\_t thread1\_attr = {

.priority = osPriorityAboveNormal //Set initial thread priority to above normal

};

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* Configure the system clock \*/

SystemClock\_Config();

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_TIM2\_Init();

/\* Init scheduler \*/

osKernelInitialize();

/\* USER CODE BEGIN RTOS\_THREADS \*/

ThreadId1 = osThreadNew(pwmThreadFunc, GPIOA,&thread1\_attr);

ThreadId2 = osThreadNew(pwmThreadFunc, GPIOB, NULL);

TimerId1 = osTimerNew(timerCallback, osTimerPeriodic, GPIOA, NULL);

TimerId2 = osTimerNew(timerCallback, osTimerPeriodic, GPIOB, NULL);

osTimerStart(TimerId1, 15U);

osTimerStart(TimerId2, 15U);

/\* USER CODE END RTOS\_THREADS \*/

/\* Start scheduler \*/

osKernelStart();

/\* We should never get here as control is now taken by the scheduler \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

}

/\* USER CODE END 3 \*/

}

void \_delay\_ms(uint32\_t ms)

{

uint32\_t start = ticks\_delay;

while((ticks\_delay - start) < ms);

}

void PWM(uint16\_t dutyCycle, uint16\_t period, GPIO\_TypeDef\* port, uint16\_t pin)

{

HAL\_GPIO\_WritePin(port, pin, GPIO\_PIN\_SET);

\_delay\_ms(dutyCycle);

HAL\_GPIO\_WritePin(port, pin, GPIO\_PIN\_RESET);

\_delay\_ms(period - dutyCycle);

}

void pwmThreadFunc(void\* argument)

{

while(1){

PWM(15, 30, argument, GPIO\_PIN\_0);

}

}

void timerCallback(void \*argument)

{

HAL\_GPIO\_TogglePin(argument, GPIO\_PIN\_8);

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the CPU, AHB and APB busses clocks

\*/

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

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

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

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

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

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB busses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

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

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = SystemCoreClock/10000U - 1;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 9;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

HAL\_TIM\_Base\_Start\_IT(&htim2);

/\* USER CODE END TIM2\_Init 2 \*/

}

/\*\*

\* @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\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0|GPIO\_PIN\_8, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0|GPIO\_PIN\_8, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : PA0 PA8 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|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 pins : PB0 PB8 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|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(GPIOB, &GPIO\_InitStruct);

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief Period elapsed callback in non blocking mode

\* @note This function is called when TIM1 interrupt took place, inside

\* HAL\_TIM\_IRQHandler(). It makes a direct call to HAL\_IncTick() to increment

\* a global variable "uwTick" used as application time base.

\* @param htim : TIM handle

\* @retval None

\*/

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

/\* USER CODE BEGIN Callback 0 \*/

/\* USER CODE END Callback 0 \*/

if (htim->Instance == TIM1) {

HAL\_IncTick();

}

/\* USER CODE BEGIN Callback 1 \*/

/\* USER CODE END Callback 1 \*/

}

/\*\*

\* @brief This function handles TIM2 global interrupt.

\*/

void TIM2\_IRQHandler(void)

{

/\* USER CODE BEGIN TIM2\_IRQn 0 \*/

++ticks\_delay; //increment counter

/\* USER CODE END TIM2\_IRQn 0 \*/

HAL\_TIM\_IRQHandler(&htim2); //pass htim2 address to handler

/\* USER CODE BEGIN TIM2\_IRQn 1 \*/

/\* USER CODE END TIM2\_IRQn 1 \*/

}

/\*\*

\* @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 \*/

/\* USER CODE END Error\_Handler\_Debug \*/

}

#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,

tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*\*/

**Висновок**: у ході виконання лабораторної роботи ми створили проект у середовищі CubeMX, і відредагували його код в IDE Keil. Ми дослідили управління затримками часу у RTOS для формування імпульсного сигналу довільної форми з використанням віртуального таймеру. Ми побудували програму для формування імпульсного сигналу з використанням RTOS і протестували механізми налагодження і управління потоками з використанням середовища MDK Keil на основі ядра CMSIS. Після написання інструкцій, ми записали програму до Flash МК та протестували її на платі налагодження STM32F103C8.