



Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

## **Лабораторна робота №1**

### **Мікропроцесорні технології інтернету речей**

**Тема:** Ознайомлення з програмою Proteus. Бібліотека HAL. Налаштування периферії за допомогою Cube MX. Апаратний ШІМ.

Виконав

студент групи ІП-11:

Панченко С. В.

Перевірив:

Стельмах А.А.

ЗМІСТ

1 Мета лабораторної роботи.....6

2 Завдання.....7

3 Виконання.....8

4 Висновок..... 13

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ..... 14

## 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Симуляція роботи мікропроцесора в програмі Proteus. Генерація коду за допомогою програми Cube MX. Реалізація ШІМ. Створення і компіляція робочої програми на мові програмування C++.

## 2 ЗАВДАННЯ

$N$  – номер студента у списку, якщо номер  $> 15$ , то  $N = N - 15$  (наприклад, якщо номер студента 16, то  $N = 16 - 15 = 1$ ;  $17 - 2$ ;  $18 - 3...$ )

$$N1 = N - 1$$

$$T1 = N / 10$$

Щоб отримати 60

1. Налаштувати тактову частоту мікроконтролера (HCLK) на  $N$  mhz
2. Підключити 10 світлодіодів та 2 кнопки до будь яких вільних пінів.
3. Якщо перша кнопка натиснута, то «активні» перші 5 світлодіодів, якщо ні то з 6-10

Щоб отримати 85

1. Виконати всі завдання з «Щоб отримати 60»
2. Змінювати сигнали з 0 на 1 та з 1 на 0 на «активних» світлодіодах, за допомогою будь якого таймеру з швидкістю  $T1$  разів за секунду.

Щоб отримати 100

1. Виконати всі завдання з «Щоб отримати 85»
2. Підключити віртуальний осцилограф та перевірити правильність виконання попереднього завдання
3. Налаштувати апаратний ШІМ на будь якому таймері та каналі використовуючи звичайний та комплементарний виводи.

### 3 ВИКОНАННЯ

Відповідно до наданого завдання маємо наступні вхідні параметри:

1.  $N = 3$
2.  $N1 = 2$
3.  $T1 = 0.3$

В якості мікроконтролера використовується модуль для макетування на рисунку 3.1.



Рисунок 3.1 — STM32F103C8T6

З наступними характеристиками:

1. Архітектура: Arm Cortex-M3
2. Флеш-пам'ять: 64Kb
3. CPU: 72MHz

Відповідно до варіанта налаштовано тактову частоту мікроконтролера (HCLK) на 3 МГц на рисунку 3.2. Початкова частота (SI) становить 8 МГц. За допомогою PLL множника, який дорівнює 3, ми збільшуємо цю частоту до 24 МГц. Потім застосовуємо дільник АНВ (HCLK prescaler), який дорівнює 8, і отримуємо кінцеву частоту 3 МГц (24 МГц поділити на 8). Таким чином, HCLK (тактова частота шини) дорівнює 3 МГц.

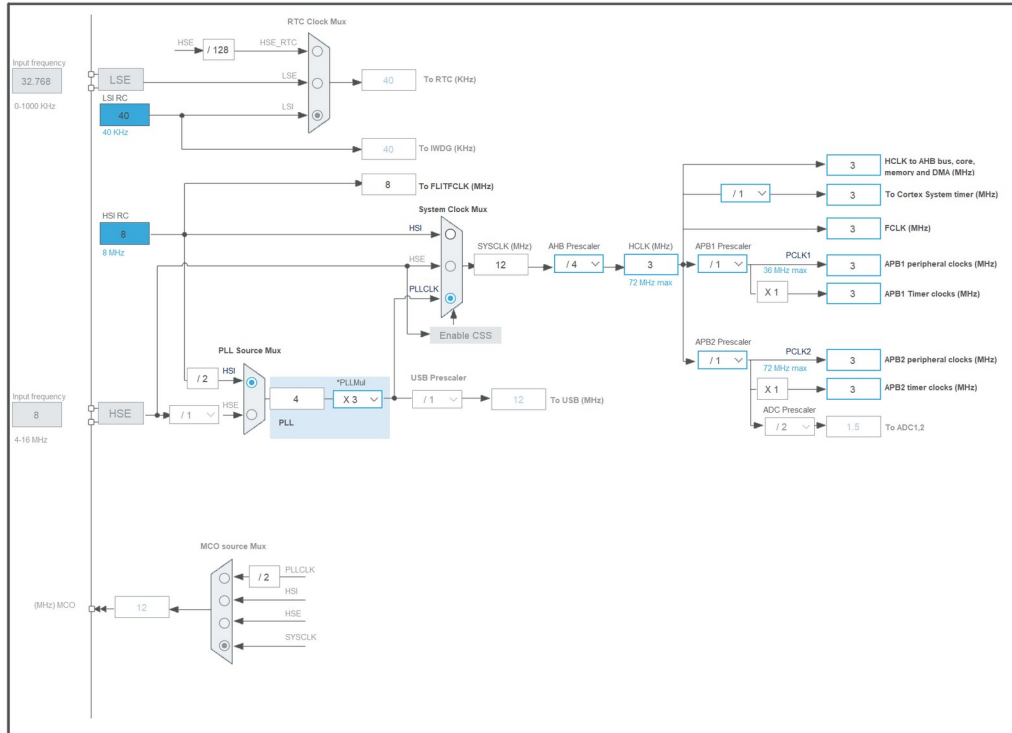


Рисунок 3.2 — налаштування тактової частоти контролера

Кнопки приєднані до пінів PB10 та PB11 у режимі PULL-UP (без додаткових резисторів). Це означає, що за замовчуванням стоїть високий рівень сигнали, і при натиску стає низький рівень сигналу. До кожного світодіоду під'єднаний струмообмежувальний резистор величиною 100 Ом.

Підключення компонентів відповідно до завдання виконано наступним чином на рисунку 3.3:

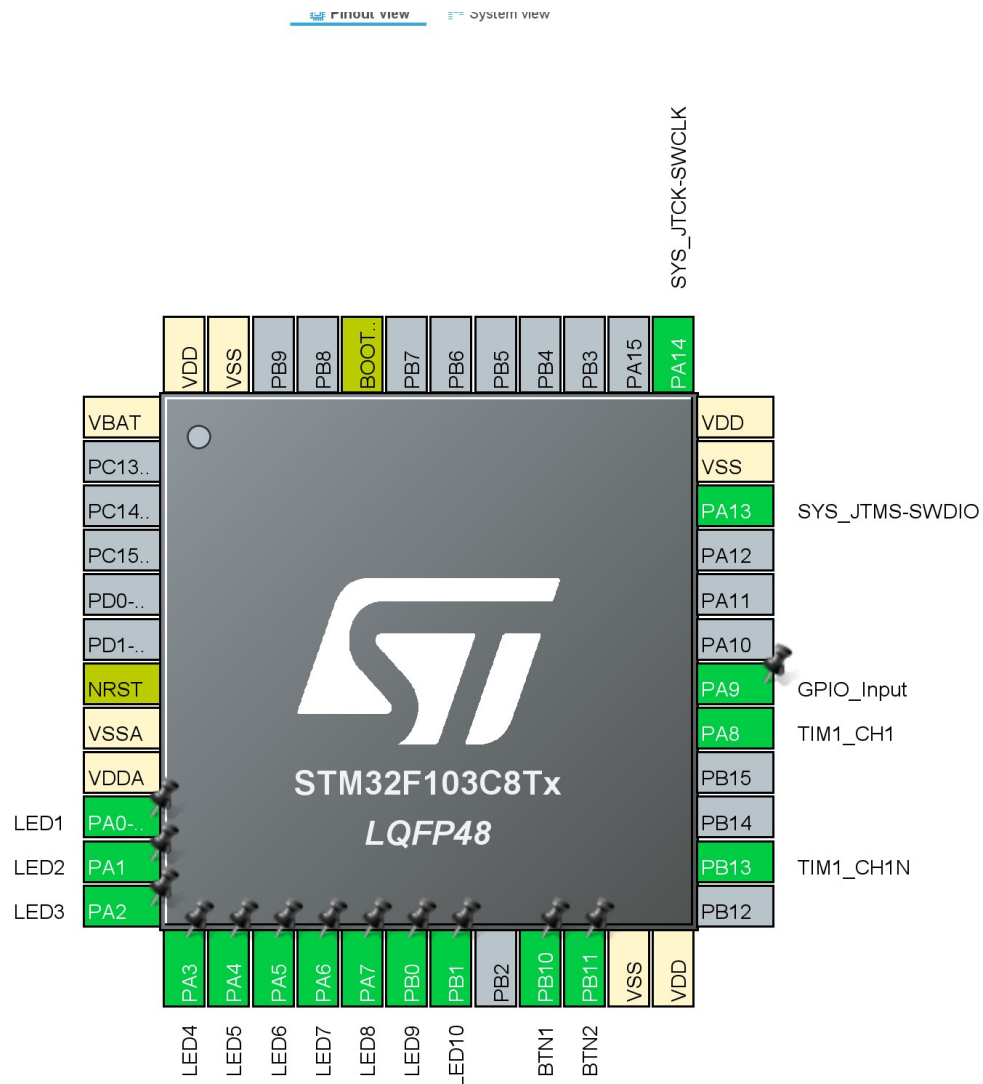


Рисунок 3.3 — конфігурація пінів (LED1: PA0, LED2: PA1, LED3: PA2, LED4: PA3, LED5: PA4, LED6: PA5, LED7: PA6, LED8: PA7, LED9: PB0, LED10: PB1, Button1: PB10, Button2: PB11)

Також відповідно до завдання налаштовано два канали апаратного таймера: звичайни та комплементарний.

Щоб отримати період 3,33 секунди з тактовою частотою 3 МГц, встановлюємо Prescaler (переддільник) на значення 9999. Це дає нам частоту після переддільника 1000 Гц, що відповідає 1 мс (3 МГц поділити на 3000). Далі встановлюємо Period (період) на значення 3329, що дає нам період 3330 мс, що приблизно дорівнює 3,33 секунди.

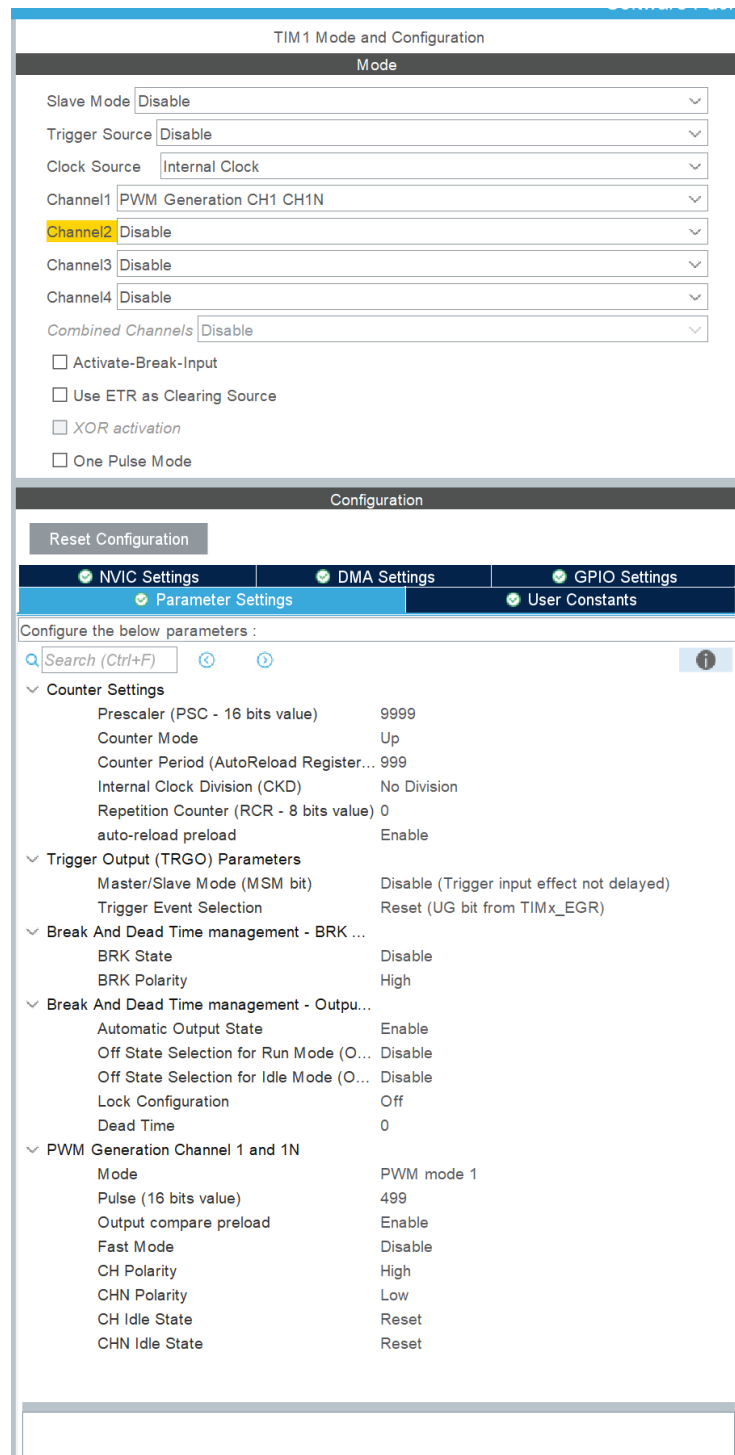


Рисунок 3.4 — конфігурація таймера

Для налаштування PWM (широотно-імпульсної модуляції) з частотою 1



кГц при тактовій частоті таймера 3 МГц спочатку встановлюємо Prescaler на значення 2. Це дає нам частоту після переддільника 1 МГц (3 МГц поділити на 3). Потім встановлюємо Period на значення 999, що дає нам остаточну частоту PWM 1 кГц (1 МГц поділити на 1000). Значення Pulse встановлюємо на 500, що забезпечує 50% робочий цикл (скважність сигналу).

Також у цій конфігурації NVIC переривання увімкнені, що дозволяє обробляти події таймера через переривання.

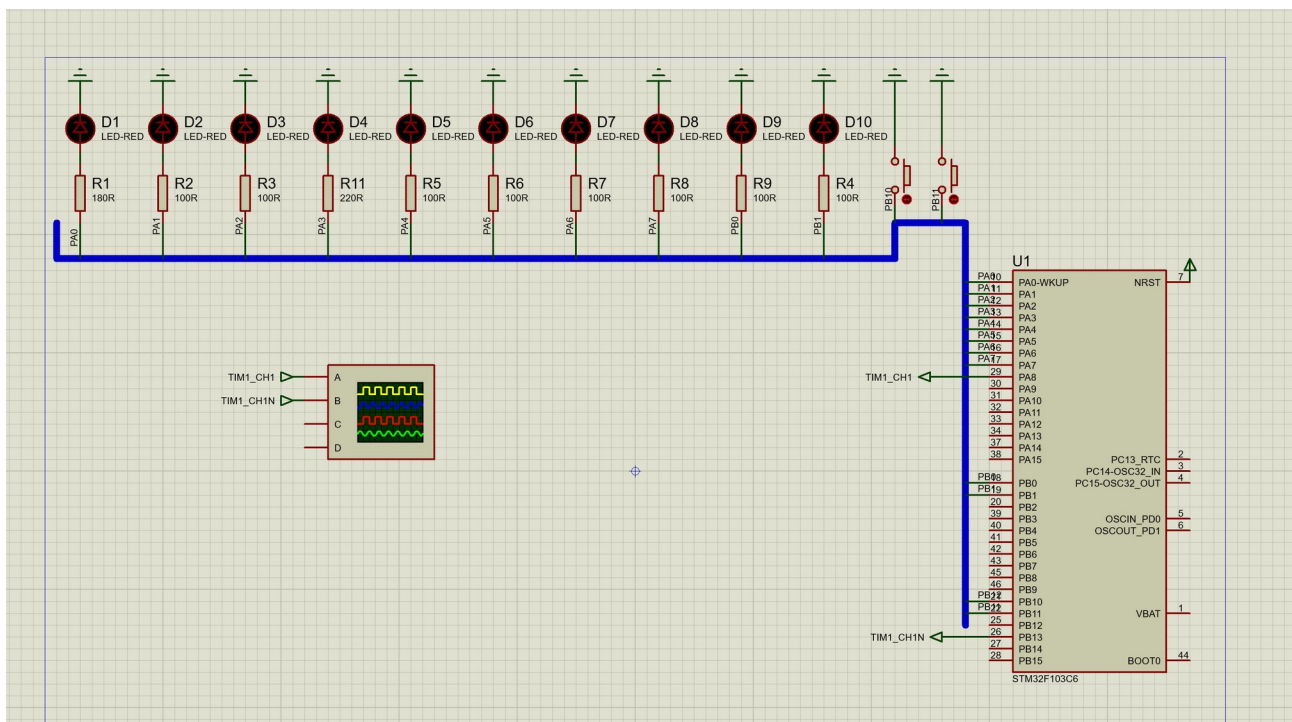


Рисунок 3.5 — схема підключення у середовищі розробки Proteus

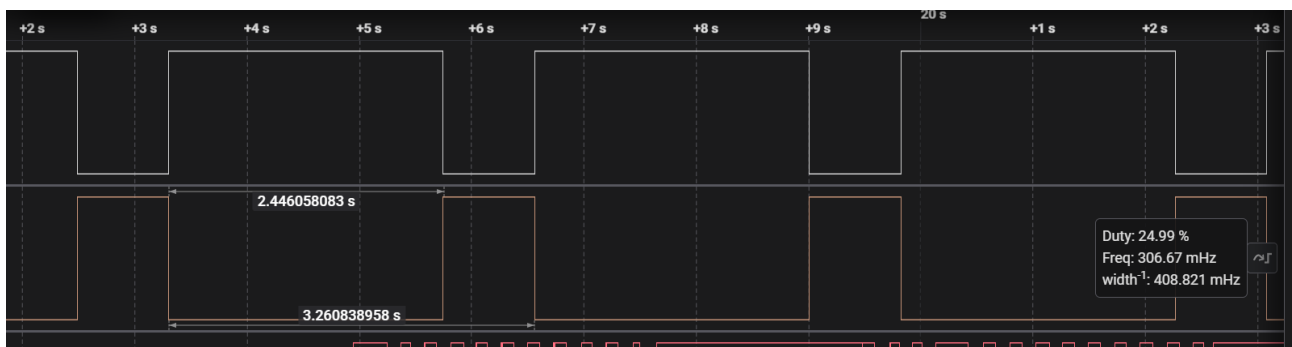


Рисунок 3.6 — показники з осцилографа

## 4 ВИСНОВОК

У ході виконання лабораторної роботи я ознайомився з можливостями програми Proteus для симуляції роботи мікропроцесорних систем, бібліотекою HAL, що забезпечує апаратну абстракцію, та інструментом STM32CubeMX для налаштування периферії мікроконтролера STM32F103C8T6. Навчився налаштовувати тактову частоту мікроконтролера на 3 МГц, конфігурувати піни для підключення світлодіодів і кнопок, а також реалізовувати апаратний ШІМ з частотою 1 кГц та скважністю 50%. Впевнився у коректності виконаної роботи шляхом перевірки параметрів сигналу на осцилографі, що підтвердило правильність розрахунків налаштування таймерів та роботи схеми з комплементарними виводами ШІМ, а також коректність перемикання активних світлодіодів при натисканні кнопок.

## ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

*Тексти програмного коду*  
(Найменування програми (документа))

*Жорсткий диск*  
(Вид носія даних)

(Обсяг програми (документа), арк.)

*Студента групи ІП-11 4 курсу*  
*Панченка С. В*

---

```
/* USER CODE BEGIN Header */
/**

*****
*****

* @file           : main.c
* @brief          : Main program body

*****
*****

* @attention

*
* Copyright (c) 2025 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 */
/* Includes
-----
*/
#include "main.h"

/* Private includes
-----*/
/* USER CODE BEGIN Includes */
#include "stm32f1xx_hal_tim.h"

/* USER CODE END Includes */
```

---

---

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

/* USER CODE END PTD */

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

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables
-----*/
TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
-----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

---

---

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

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
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_TIM1_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
//      GPIO_PinState btn1_state = HAL_GPIO_ReadPin(GPIOB,
BTN1_Pin);
//      GPIO_PinState btn2_state = HAL_GPIO_ReadPin(GPIOB,
BTN2_Pin);
//
//      HAL_GPIO_WritePin(GPIOA, LED1_Pin | LED2_Pin | LED3_Pin |
LED4_Pin | LED5_Pin,
//
//                      (btn1_state == GPIO_PIN_RESET) ?
GPIO_PIN_SET : GPIO_PIN_RESET);
//
//      HAL_GPIO_WritePin(GPIOA, LED6_Pin | LED7_Pin | LED8_Pin,
//
//                      (btn2_state == GPIO_PIN_RESET) ?
GPIO_PIN_SET : GPIO_PIN_RESET);
//
//      HAL_GPIO_WritePin(GPIOB, LED9_Pin | LED10_Pin,
//
//                      (btn2_state == GPIO_PIN_RESET) ?
GPIO_PIN_SET : GPIO_PIN_RESET);

      GPIO_PinState pwm_level = HAL_GPIO_ReadPin(GPIOA,
GPIO_PIN_9);

      // button state
      GPIO_PinState btn1_state = HAL_GPIO_ReadPin(GPIOB,
BTN1_Pin);
      GPIO_PinState btn2_state = HAL_GPIO_ReadPin(GPIOB,
BTN2_Pin);
```

---

---

```

        // BTN1 – LED1-LED5
        HAL_GPIO_WritePin(GPIOA,
                           LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin
| LED5_Pin,
                           (btn1_state == GPIO_PIN_RESET) ? pwm_level
: GPIO_PIN_RESET);

        // BTN2 – LED6-LED10
        HAL_GPIO_WritePin(GPIOA,
                           LED6_Pin | LED7_Pin | LED8_Pin,
                           (btn2_state == GPIO_PIN_RESET) ? pwm_level
: GPIO_PIN_RESET);

        // BTN2 – LED6-LED10
        HAL_GPIO_WritePin(GPIOB,
                           LED9_Pin | LED10_Pin,
                           (btn2_state == GPIO_PIN_RESET) ? pwm_level
: GPIO_PIN_RESET);

        /* USER CODE END WHILE */

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

/**
 * @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.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL3;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
RCC_CLOCKTYPE_SYCLK
                                |RCC_CLOCKTYPE_PCLK1|
RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV4;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
HAL_OK)
    {
        Error_Handler();
    }
}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None

```

---

---

```
*/
static void MX_TIM1_Init(void)
{

    /* USER CODE BEGIN TIM1_Init 0 */

    /* USER CODE END TIM1_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {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 = 9999;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 999;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) !=
    HAL_OK)
    {
        Error_Handler();
    }
    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 = 250;
    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();
    }

    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;
    sBreakDeadTimeConfigAutomaticOutput =
TIM_AUTOMATICOUTPUT_ENABLE;
    if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig)
!= HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM1_Init 2 */
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    Starting PWM

```

---

---

```

    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);

    __HAL_TIM_SET_COUNTER(&htim1,
0);                                // Counter Reset
    __HAL_TIM_ENABLE_IT(&htim1,
TIM_IT_UPDATE);                    // Interrupt Enable

__HAL_TIM_MOE_ENABLE(&htim1);      //
MANDATORY

__HAL_TIM_ENABLE(&htim1);          //
Start the timer
    HAL_TIM_GenerateEvent(&htim1, TIM_EVENTSOURCE_UPDATE); //
Generate the event
    /* USER CODE END TIM1_Init 2 */
    HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

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

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, LED1_Pin|LED2_Pin|LED3_Pin|LED4_Pin
|LED5_Pin|LED6_Pin|LED7_Pin|LED8_Pin,

```

---

---

```
GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, LED9_Pin|LED10_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LED1_Pin LED2_Pin LED3_Pin LED4_Pin
                        LED5_Pin LED6_Pin LED7_Pin LED8_Pin */
GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin|LED3_Pin|LED4_Pin
                    |LED5_Pin|LED6_Pin|LED7_Pin|LED8_Pin;
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 : LED9_Pin LED10_Pin */
GPIO_InitStruct.Pin = LED9_Pin|LED10_Pin;
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 : BTN1_Pin BTN2_Pin */
GPIO_InitStruct.Pin = BTN1_Pin|BTN2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : PA9 */
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/*Configure GPIO pin : PB13 as TIM1_CH1N (Alternate Function
Push-Pull)*/
GPIO_InitStruct.Pin = GPIO_PIN_13;
```

---

---

```

    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @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 */
}

#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 */
```

---