Ćwiczenie 1

PROGRAMOWANIE SYSTEMÓW WBUDOWANYCH

Obsługa linii cyfrowych GPIO mikrokontrolera Cortex-M4 STM32F303RE lub STM32F411RE.

Celem ćwiczenia jest zapoznanie studenta z metodyką programowania, projektowania i tworzenia aplikacji na mikrokontrolery rodziny ARM Cortex-M4,. W trakcie ćwiczenia student nabędzie podstawowe informacje dotyczące środowiska i umiejętności posługiwania się nim oraz programowania linii cyfrowych ogólnego zastosowania - GIPO.

Zakład Systemów Informacyjno-Pomiarowych

IETiSIP, Wydział Elektryczny, PW

Ćwiczenie 1 polegać będzie na dostępie i zmianie stanów linii cyfrowych. Niech składa się z kilku podzadań:

1. Pojedyncza zmiana stanu linii cyfrowej.
2. Cykliczna zmiana stanu linii cyfrowej – polling.
    a. Funkcja `HAL_GPIO_Write`
    b. Funkcja `HAL_GPIO_Toggle`
3. Zmiana czasu cyklicznej modyfikacji stanu linii – polling.
    a. Funkcja `HAL_GPIO_Read` – odczyt przycisku
4. Cykliczna zmiana stanu linii cyfrowej – przerwania (reakcja jedynie na wybraną linię.
5. Zmiana czasu cyklicznej modyfikacji stanu linii – przerwania.
    a. Funkcja `HAL_GPIO_EXTI_Callback`
6. Ustawienie i zmiana parametrów pracy, stanów linii cyfrowych
    a. Z1 - Światła na skrzyżowaniu przycisk odwrócony cykl
    b. Z2 - wędrująca jedynka przycisk odwrócony cykl
    c. Z3 - wędrujące zero przycisk odwrócony cykl
    d. Z4 - zliczanie binarne przycisk odwrócony cykl
    e. Z5 – cykl linia 1: 200ms; linia 2: 300 ms; linia 3: 600ms przycisk linia 1: 150ms; linia 2: 100 ms; linia 3: 300ms
    f. Z6 – linijka „LED”owa załączana co 500ms, przycisk 250ms
7. Podwójne naciśnięcie (w ciągu do 0.5s – zatrzymanie)

Przykładowe wywołania funkcji (użyte własne etykiety):

```
HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
GPIOx – oznaczenie portu
GPIO_Pin – oznaczenie pinu/linii na porcie
GPIO_PinState – stan linii
```

Przykładowe wywołania funkcji:

```
HAL_GPIO_WritePin(GPIOA, GreenLED_Pin , GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5 , GPIO_PIN_RESET);

GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
GPIO_PinState – typ wartości zwracanej
GPIOx – oznaczenie portu
GPIO_Pin – oznaczenie pinu/linii na porcie
```

Przykładowe wywołanie funkcji (należy pamiętać o definicji wartości zwracanej):

```
stanPinu = HAL_GPIO_ReadPin(GPIOC, UserButton_Pin);
stanPinu = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);

HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
GPIOx – oznaczenie portu
```

`GPIO_Pin` – oznaczenie pinu/linii na porcie

Przykładowe wywołanie funkcji:

```
HAL_GPIO_TogglePin(GPIOA, GreenLED_Pin);
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
```

```
HAL_Delay(uint32_t Delay)
Delay – opóźnienie w ms.
```

Przykładowe wywołanie funkcji:

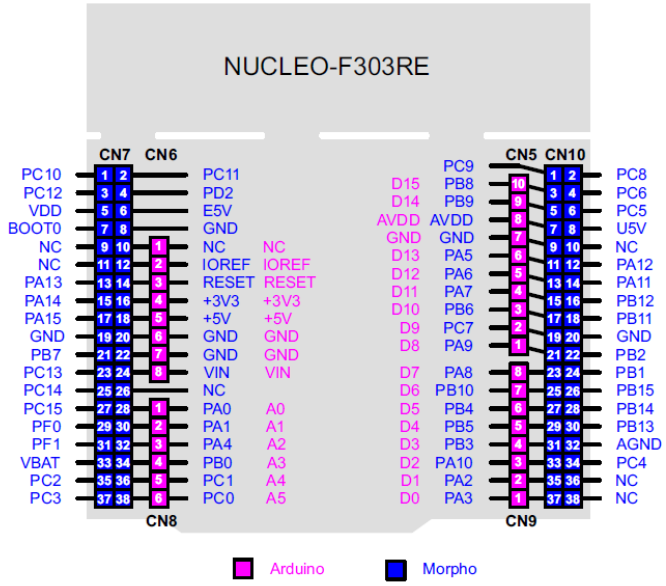
```
HAL_Delay(1000);
```

```
uint32_t HAL_GetTick(void);
```

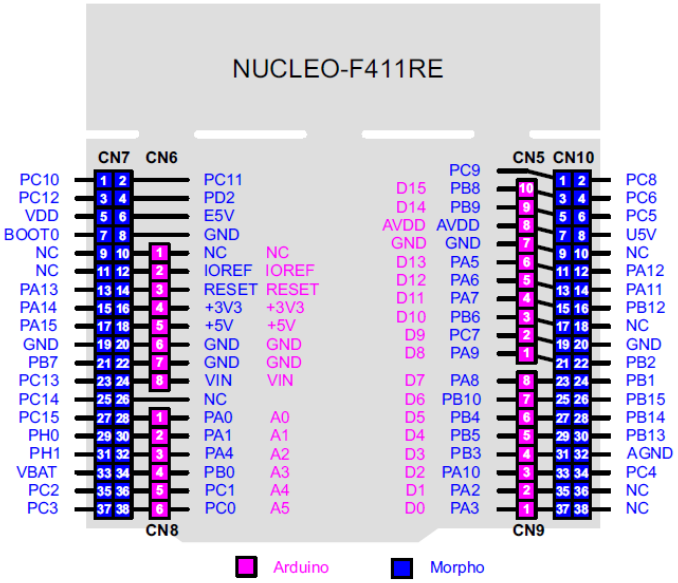
Przykładowe wywołanie funkcji (należy pamiętać o definicji wartości zwracanej):

```
czas = HAL_GetTick();
```

Szczegółowy opis funkcji znajduje się na końcu instrukcji.

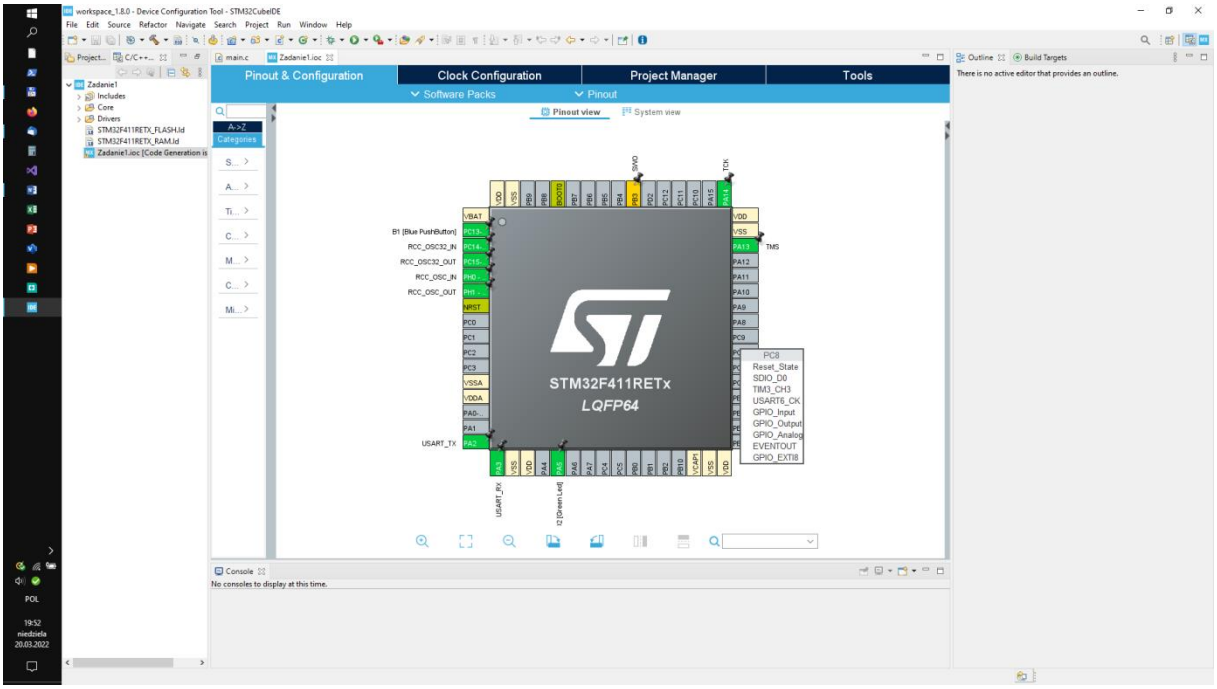


Rysunek 8. Wyprowadzenia na płytce NUCLEO-F303RE. Źródło: STM32 Nucleo-64 boards (MB1136.pdf).



Rysunek 9. Wyprowadzenia na płytce NUCLEO-F411RE. Źródło: STM32 Nucleo-64 boards (MB1136.pdf).

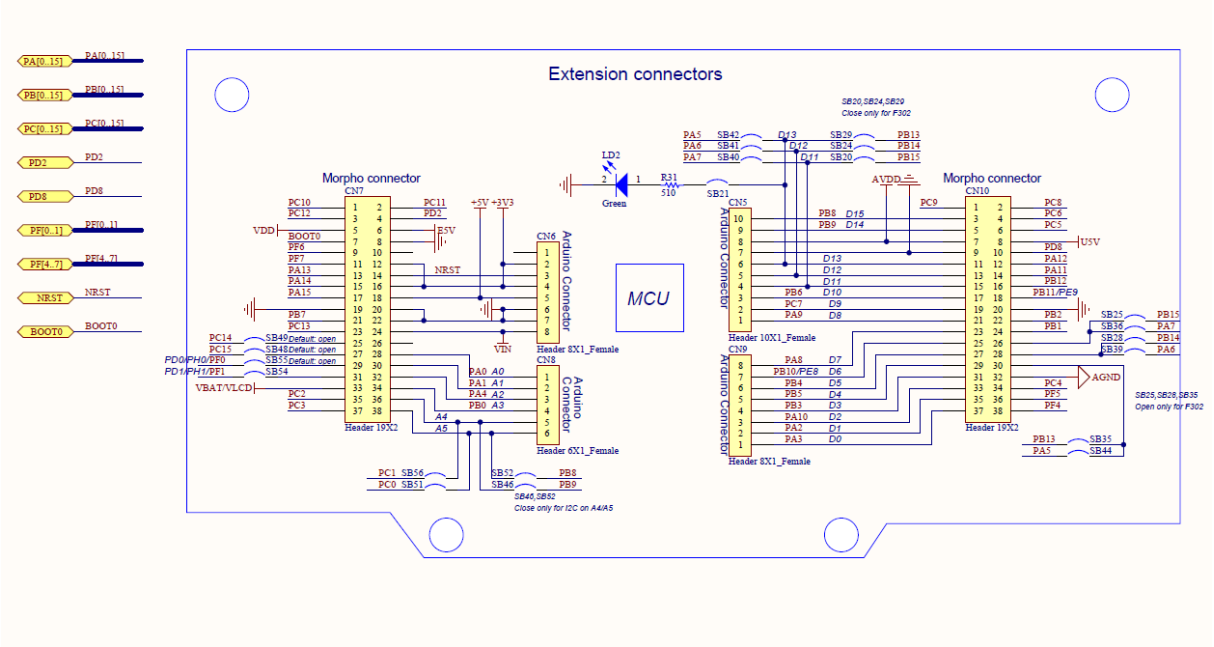
Rysunki 8 i 9 przedstawiają wyprowadzenia dla płytek NUCLEO dla dwóch mikrokontrolerów STM23F411RE i STM23F4303RE. Dla wymienionych mikrokontrolerów przykładowe dostępne linie cyfrowe to: PC5, PC6, PC8, LD2->PA5.



Rysunek 10. Konfiguracja linii cyfrowych NUCLEO-F411RE.

Rysunek 10 przedstawia możliwości nastaw dla linii PC8. Te podstawowe i często wykorzystywane to **GPIO_Input** (wejście), **GPIO_Output** (wyjście) i

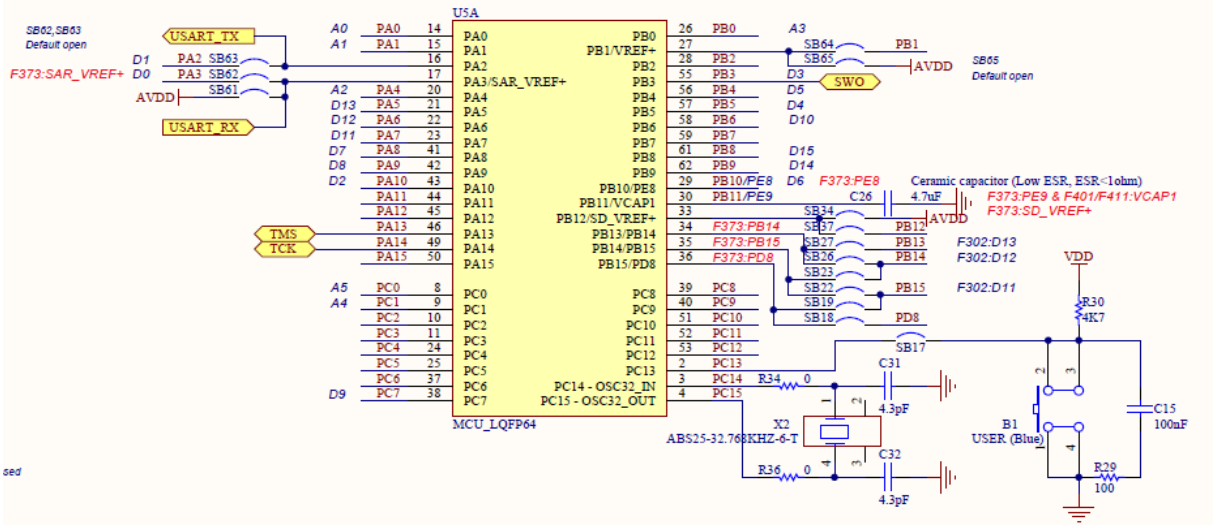
GPIO_EXTI8 (przerwanie). Możliwe jest wprowadzanie własnych etykiet dla wyprowadzeń (Nazwy własne bez spacji i polskich znaków diakrytycznych).



Rysunek 11. Połączenia elektryczne NUCLEO-F411RE – dioda LD2 Green. Źródło: STM32 Nucleo-64 boards (MB1136.pdf).

Na rysunku 11 przedstawiony został schemat elektryczny płytki NUCLEO, na którym widać, połączenie zielonej diody LED (LD2).

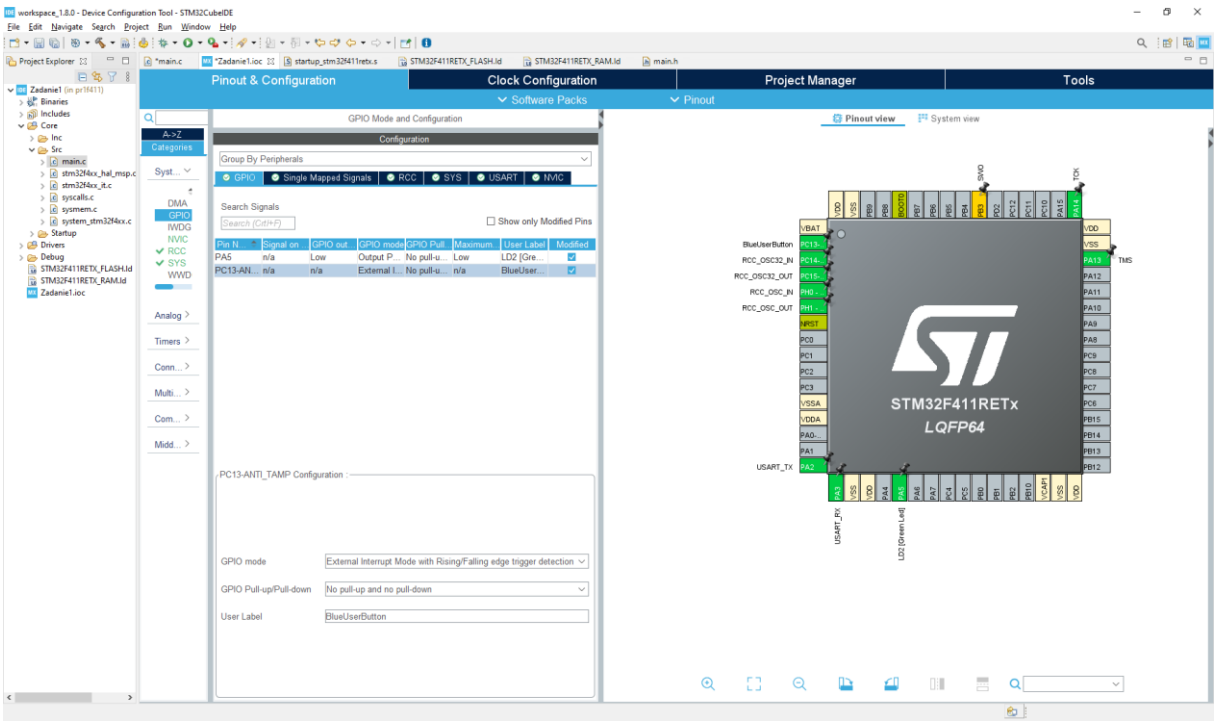
Odczyt przycisku:



Rysunek 12. Połączenia elektryczne NUCLEO-F411RE – przycisk B1 USER (Blue). Źródło: STM32 Nucleo-64 boards (MB1136.pdf).

Przycisk użytkownika podłączony jest do linii PC13->User Button (BLUE) i podciągnięty do zasiania - fizyczny pull up (R30). Proszę zwrócić uwagę na elementy C15 i R29. Stanowią one filtr minimalizujący efekt drgań mechanicznych styków przycisku. Układ ten nie występuje wszędzie (np.: inne płytki STM32) i wtedy należy go samodzielnie dolutować albo zapewnić eliminację programową.

Przerwania. Aby zmienić sposób działania programu z typowego pollingu na wykonywanie w przerwaniach należy przejść do widoku konfiguracji sprzętowej poprzez wybór pliku *.ioc.



Rysunek 13. Konfiguracja przerwań.

Następnie poprzez **Categories**-> **System Core** -> **GPIO** należy przejść do konfiguracji linii PC13 (przycisk użytkownika) i dokonać ustawień jak na rysunku 13. W tym momencie należy zwrócić uwagę aby modyfikowane parametry linii cyfrowej potwierdzone były w kolumnie **Modified**.

GPIO mode – tryb rising/falling, gdyż zmiana trybu pracy cyklicznej powinna odbywać się jedynie w czasie wciśnięcia przycisku.

GPIO Pull-up/Pull down – wyłączona konfiguracja wewnętrzna gdyż wejście jest „podciągnięte" zewnętrznie.

User Label – własna etykieta/nazwa. Należy pamiętać aby wprowadzać nazwy bez spacji i polskich znaków diakrytycznych.

Po wprowadzeniu zmian i wygenerowaniu kodu dodana jest funkcja obsługi przerwania. Jest ona umieszczona w pliku źródłowym zawierającym funkcje HAL dla GPIO. Można ją odszukać wybierając z meny Search->File.. i następnie w polu **Containing text**: wpisać **weak*callback**, pozostawiając w polu **File** *. Poszukiwana funkcja szablonowo zdefiniowana jest jako **weak**, tzn., że można (nawet zalecane jest) skopiować jej definicję do pliku źródłowego użytkownika (w zadaniu jest to main.c) już bez słowa kluczowego **weak** i tam wstawiać kod wykonawczy. Należy pamiętać o deklaracji i definicji nowej funkcji. W zadaniu nowa funkcja ma nazwę **`HAL_GPIO_EXTI_Callback`** i oryginalnie/szablonowo znajduje się w pliku stm32fxxx_hal_gpio.c

Funkcja przerwania „obsługuje" kilka linii cyfrowych zatem aby obsłużyć wyłącznie wybraną linię (w tym przypadku przycisk) w funkcji **`HAL_GPIO_EXTI_Callback`** należy dodać instrukcję warunkową np.:

```
if(GPIO_Pin == UserButton_Pin) /* GPIO_Pin jest przekazywany jako parametr do funkcji przerwania*/
```

Przydatne skróty:
Ctrl+Spacja – parametry funkcji
Ctrl+/ - komentarz
Ctrl+s – zapis
Ctrl+Shift+f – autoformatowanie

Kod głównego pliku źródłowego wygenerowany automatycznie. Należy zwracać uwagę na komentarze i własny kod wstawiać jedynie w sekcjach USER pomiędzy BEGIN a END. Nigdy odwrotnie i nigdzie indziej !

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 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 */

/* 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 ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_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
  */
```

```c
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_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* 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};
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  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_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 USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */
```

```
  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_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_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
```

```
}

/* 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 */
```

Dokumentacja funkcji użytych w programach

```
/**
  * @brief This function provides minimum delay (in milliseconds) based
  *        on variable incremented.
  * @note In the default implementation , SysTick timer is the source of time
base.
  *       It is used to generate interrupts at regular time intervals where uwTick
  *       is incremented.
  * @note This function is declared as __weak to be overwritten in case of other
```

```
  *       implementations in user file.
  * @param Delay specifies the delay time length, in milliseconds.
  * @retval None
  */
__weak void HAL_Delay(uint32_t Delay)
{
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

  /* Add a freq to guarantee minimum wait */
  if (wait < HAL_MAX_DELAY)
  {
    wait += (uint32_t)(uwTickFreq);
  }

  while((HAL_GetTick() - tickstart) < wait)
  {
  }
}
/**
  * @brief Provides a tick value in millisecond.
  * @note This function is declared as __weak to be overwritten in case of other
  *       implementations in user file.
  * @retval tick value
  */
__weak uint32_t HAL_GetTick(void)
{
  return uwTick;
}

/**
  * @brief Return tick frequency.
  * @retval tick period in Hz
  */
HAL_TickFreqTypeDef HAL_GetTickFreq(void)
{
  return uwTickFreq;
}

/**
  * @brief  Reads the specified input port pin.
  * @param  GPIOx where x can be (A..K) to select the GPIO peripheral for
STM32F429X device or
  *                      x can be (A..I) to select the GPIO peripheral for
STM32F40XX and STM32F427X devices.
  * @param  GPIO_Pin specifies the port bit to read.
  *         This parameter can be GPIO_PIN_x where x can be (0..15).
  * @retval The input port pin value.
  */
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

```
{
  GPIO_PinState bitstatus;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
  {
    bitstatus = GPIO_PIN_SET;
  }
  else
  {
    bitstatus = GPIO_PIN_RESET;
  }
  return bitstatus;
}

/**
  * @brief  Sets or clears the selected data port bit.
  *
  * @note   This function uses GPIOx_BSRR register to allow atomic read/modify
  *         accesses. In this way, there is no risk of an IRQ occurring between
  *         the read and the modify access.
  *
  * @param  GPIOx where x can be (A..K) to select the GPIO peripheral for
STM32F429X device or
  *                       x can be (A..I) to select the GPIO peripheral for
STM32F40XX and STM32F427X devices.
  * @param  GPIO_Pin specifies the port bit to be written.
  *          This parameter can be one of GPIO_PIN_x where x can be (0..15).
  * @param  PinState specifies the value to be written to the selected bit.
  *          This parameter can be one of the GPIO_PinState enum values:
  *            @arg GPIO_PIN_RESET: to clear the port pin
  *            @arg GPIO_PIN_SET: to set the port pin
  * @retval None
  */
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState
PinState)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  assert_param(IS_GPIO_PIN_ACTION(PinState));

  if(PinState != GPIO_PIN_RESET)
  {
    GPIOx->BSRR = GPIO_Pin;
  }
  else
  {
    GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
```

```
  }
}

/**
  * @brief  Toggles the specified GPIO pins.
  * @param  GPIOx Where x can be (A..K) to select the GPIO peripheral for
STM32F429X device or
  *                      x can be (A..I) to select the GPIO peripheral for
STM32F40XX and STM32F427X devices.
  * @param  GPIO_Pin Specifies the pins to be toggled.
  * @retval None
  */
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  uint32_t odr;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  /* get current Ouput Data Register value */
  odr = GPIOx->ODR;

  /* Set selected pins that were at low level, and reset ones that were high */
  GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```