Ćwiczenie 3

PROGRAMOWANIE SYSTEMÓW WBUDOWANYCH

Obsługa układów czasowo-licznikowych – timerów, 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 układów czasowo-licznikowych - timerów.

Zakład Systemów Informacyjno-Pomiarowych

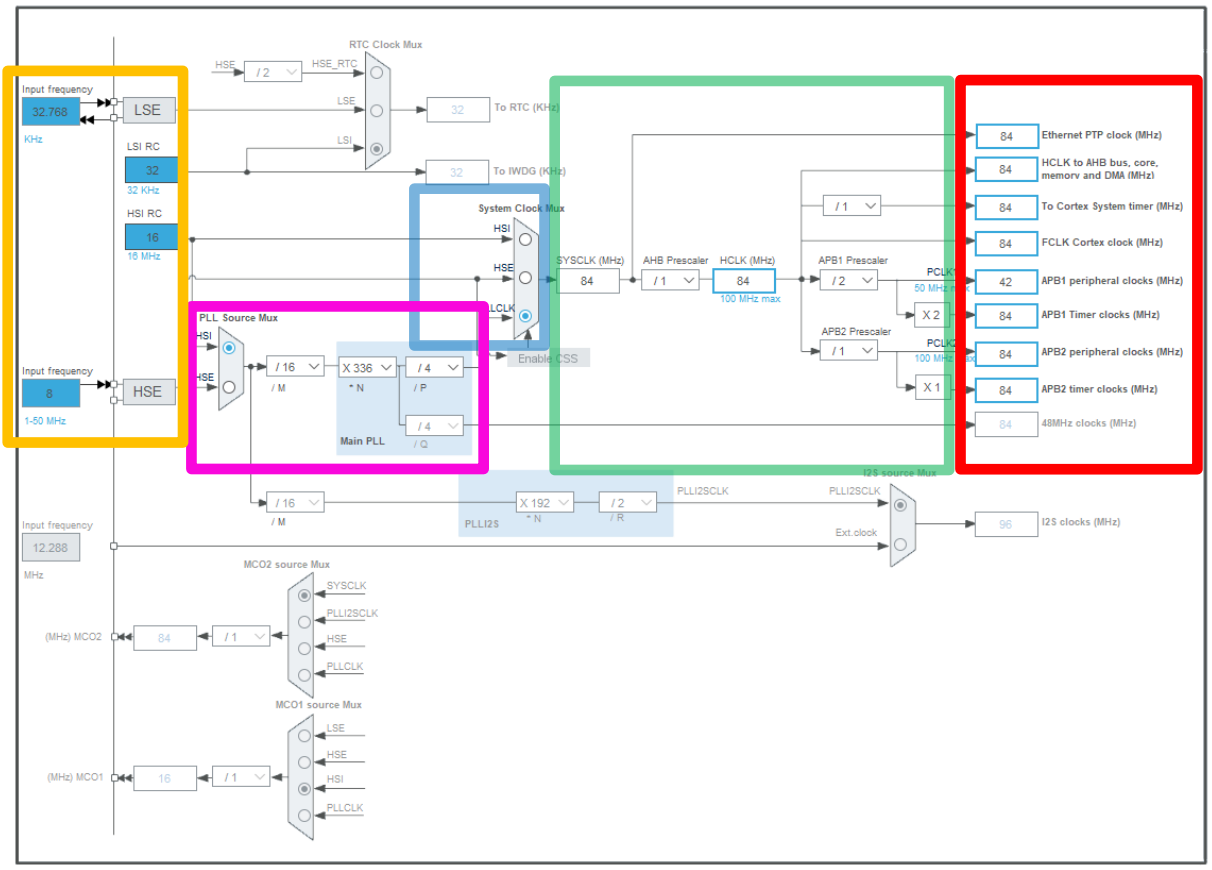
IETiSIP, Wydział Elektryczny, PW

Ćwiczenie 3 polegać będzie na konfiguracji i wykorzystania układów czasowo-licznikowych (timerów). Niech składa się z kilku podzadań:

1. Wykorzystanie funkcji `HAL_Delay`, `HAL_GetTick`
2. Konfiguracja timera .
3. Wykorzystanie timera – tryb przerwań.
    a. Z1- 3Hz
    b. Z2 – 4Hz
    c. Z3 – 5Hz
    d. Z4 - 7Hz
    e. Z5 – 9Hz
    f. Z6 – 11Hz
4. Proszę określić rozdzielczość czasową generowanych impulsów
5. Różne czasy włączenia i wyłączenia diody
    a. Z1 - Włączona: 100ms wyłączona 200ms
    b. Z2 - Włączona: 150ms wyłączona 300ms
    c. Z3 - Włączona: 200ms wyłączona 400ms
    d. Z4 - Włączona: 300ms wyłączona 600ms
    e. Z5 - Włączona: 250ms wyłączona 500ms
    f. Z6 - Włączona: 400ms wyłączona 800ms
6. „PWM”
    a. Z1 – Okres: 200ms, rozdzielczość 10%
    b. Z2 - Okres: 300ms, rozdzielczość 5%
    c. Z3 - Okres: 400ms, rozdzielczość 10%
    d. Z4 - Okres: 600ms, rozdzielczość 5%
    e. Z5 - Okres: 500ms, rozdzielczość 10%
    f. Z6 - Okres: 800ms, rozdzielczość 5%

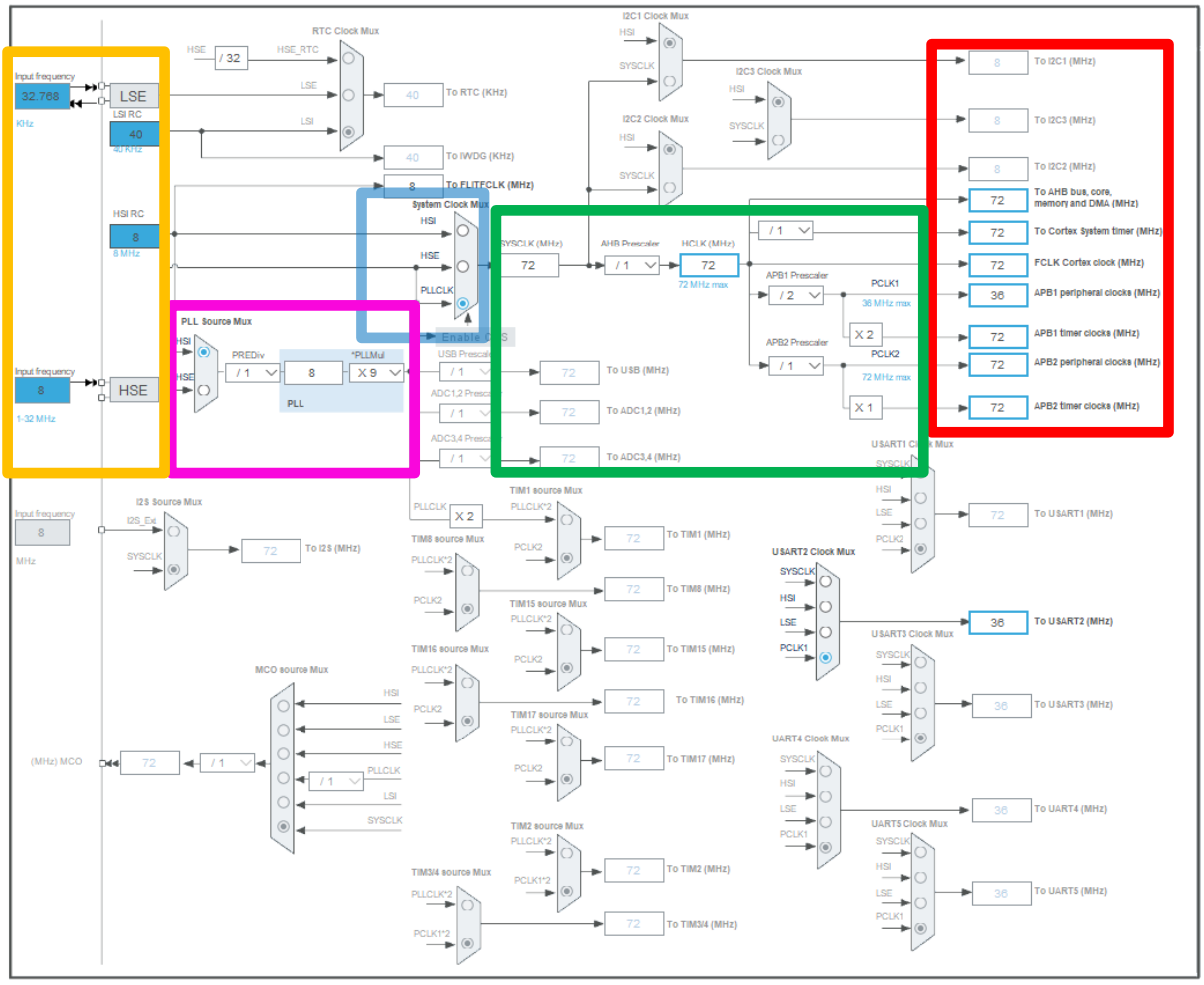
Oznaczenia i konfiguracja timerów w mikrokontrolerach STM32F303 oraz STM32F411:

Clock_Configuration (STM32F411):



Rysunek 1a. Konfiguracja linii zegarowych STM32F411

Clock_Configuration (STM32F303):



Rysunek 1b. Konfiguracja linii zegarowych STM32F303

Schemat konfiguracyjny linii zegarowych przedstawiony został na rysunku 1. Źródła zegarowych sygnałów wejściowych (oznaczone kolorem żółtym) można wybrać spośród poniższych:

- **LSE** (**Low Speed External**) – zewnętrzny oscylator małej częstotliwości (na płytce NUCLEO wynosi ona 32.768kHz
- **LSI RC** - (**Low Speed Internal**) – wewnętrzny generator małej częstotliwości RC 32/40kHz (F303/F411)
- **HSE** - (**High Speed External**) – zewnętrzny oscylator wysokiej częstotliwości (na płytce NUCLEO niepodłączony
- **HSI RC** - (**High Speed Internal**) – wewnętrzny generator RC 1-32MHz/1-50MHz (F303/F411)

Sygnał pochodzący z zewnętrznego oscylatora charakteryzuje się większą stabilnością w porównaniu do tego z generatora wewnętrznego RC.

PLLCLK (Phase Locked Loop Clock) – oznaczona kolorem fioletowym na rysunku 1 blok/układ pętli fazowej umożliwiający zwielokrotnienie częstotliwości sygnału generowanego w układzie HSI lub HSE. Blok ten można konfigurować (źródło/mnożnik) w sposób określony również w sekcji fioletowej.

Na rysunku 1 kolorem niebieskim oznaczono blok wyboru źródła sygnału taktującego, którego wyjście przekazywane jest do układów mnożących i dzielących (oznaczonych kolorem zielonym), które pozwalają na ostateczne ustawienie częstotliwości sygnałów taktujących dla poszczególnych modułów MCU (sekcja czerwona na rysunku 1).

W sekcji tej wyróżnić można:

- **FCLK** - zegar główny mikrokontrolera. Dla mikrokontrolera STM32F411 wartość maksymalna wynosi 100MHz natomiast dla mikrokontrolera STM32F303 wynosi ona 72MHz.
- **APB1** – magistrala wolna, (low-speed). Do nie j podłączone są peryferia: I2C, część układów USART, SPI, I2S oraz część timerów. Peryferia (za wyjątkiem timerów podłączonych APB1 Timer Clocks) podłączone do tej magistrali mogą być taktowane maksymalnie 50MHZ (36MHz)
- **APB2** - magistrala szybka, (high-speed). Dostarcza taktowanie dla pozostałych peryferiów SPI, SPI, SPI, ADC, USART oraz dla timerów. Maksymalna częstotliwość na tej magistrali, to 100MHz/72MHz**.**

Każda z wymienionych magistrali dysponuje dwoma magistralami podrzędnymi. Jedna z wymienionych zapewnia sygnał zegarowy peryferiom - Peripheral Clocks, a druga timerom - Timer Clocks.

Będące na wyposażeniu mikrokontrolerów STM32 timery są zaawansowanymi układami peryferyjnymi, mającymi wszechstronne zastosowanie w licznych aplikacjach. I tak mogą być wykorzystane przy:

- pomiarach czasu,
- zliczaniu impulsów/zdarzeń,
- generowaniu „bramek” czasowych
- generowaniu przerwań jednorazowo i cyklicznie,
- modulowaniu sygnału PWM,
- obsłudze enkoderów kwadraturowych,
- rozkodowywanie sygnału PWM,
- współpracy z wieloma czujnikami.

Przykładowe wywołania funkcji (niektóre nazwy są własne):

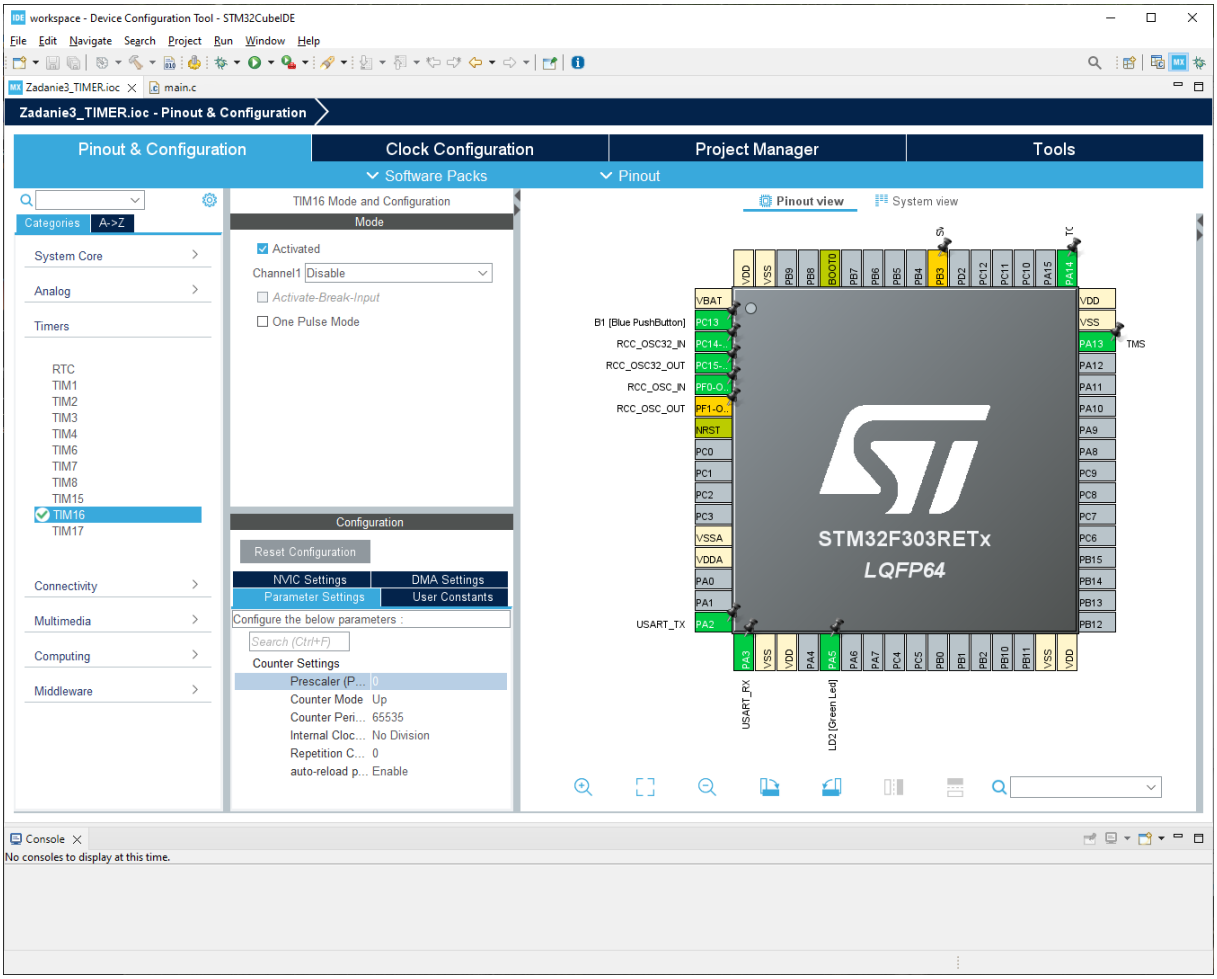
```
MX_TIM16_Init();/*inicjacja modułu timerów*/
HAL_TIM_Base_Start_IT(&htim16); /*aktywacja mechanizmu przerwań dla
TIM16*/
```

```
HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim); /*funkcja obsługi przerwania*/
HAL_GPIO_TogglePin(GPIOA, GreenLed_Pin); /*Ustawianie linii cyfrowej*/
```

Szczegółowy opis wszystkich funkcji znajduje się na końcu instrukcji.

Należy stworzyć nowy projekt (**File->New->STM32 Project**) i wybrać plik *.ioc. Następnie przejść do okna **Clock Configuration** i sprawdzić czy Magistrale **APB** zasilające timery są ustawione na częstotliwość 72MHz (STM32F303) lub 100MHz (STM32F411). W konfiguracji STM32F303 należy sprawdzić przede wszystkim wartość częstotliwości na wyjściu **TIM16** (72MHz na rysunku 1b).
1.Konfiguracja timera.

W oknie przedstawionym na rysunku 2 należy przejść do **Categories-> Timers->TIM16** (STM32F303) lub **Categories-> Timers->TIM10** (STM32F411) i zaznaczyć opcję: **Activated**. Niezależnie trzeba sprawdzić ustawienia linii PA5 (Zielona dioda LED) tak aby można nią było sterować.



Rysunek 2. Konfiguracja TIMERa TIM16 – STM32F303.

W oknie **Parameters Settings** pojawiają się następujące opcje:

- **Prescaler (PSC)** – wewnętrzny podzielnik zegara taktującego timer wartość (rejestr podzielnika jest 16-bitowy zatem maksymalna wartość wynosi 65535).
- **Counter Mode** – timer zlicza „w górę" albo „w dół".
- **Counter Period (Auto Reload Register - ARR)** - wartość, do której zlicza timer.
- **Internal Clock Division (CKD)** - jeszcze jedno miejsce, gdzie można dokonać dzielenia sygnału taktującego timer.
- **Repetition Counter** – licznik cyklów zliczeń **ARR**(rejestr 8 bitowy)
- **auto-reload preload** – aktywowanie automatycznego przeładowania licznika.

Zasada działania licznika:

1. Wartość początkowa timera wynosi 0 (licznik w górę).
2. Każdy impuls zegara taktującego zwiększa wartość rejestru licznika o jeden. Częstotliwość taktującą timer określa zawartość rejestrów (podzielnikowych) **PSC** oraz **ARR**.
3. W chwili gdy zawartość timera zrówna się z wartością rejestru ARR, timer generuje przerwanie, ustawiana jest wartość początkowa rejestru i czynność zliczania rozpoczyna się na nowo.

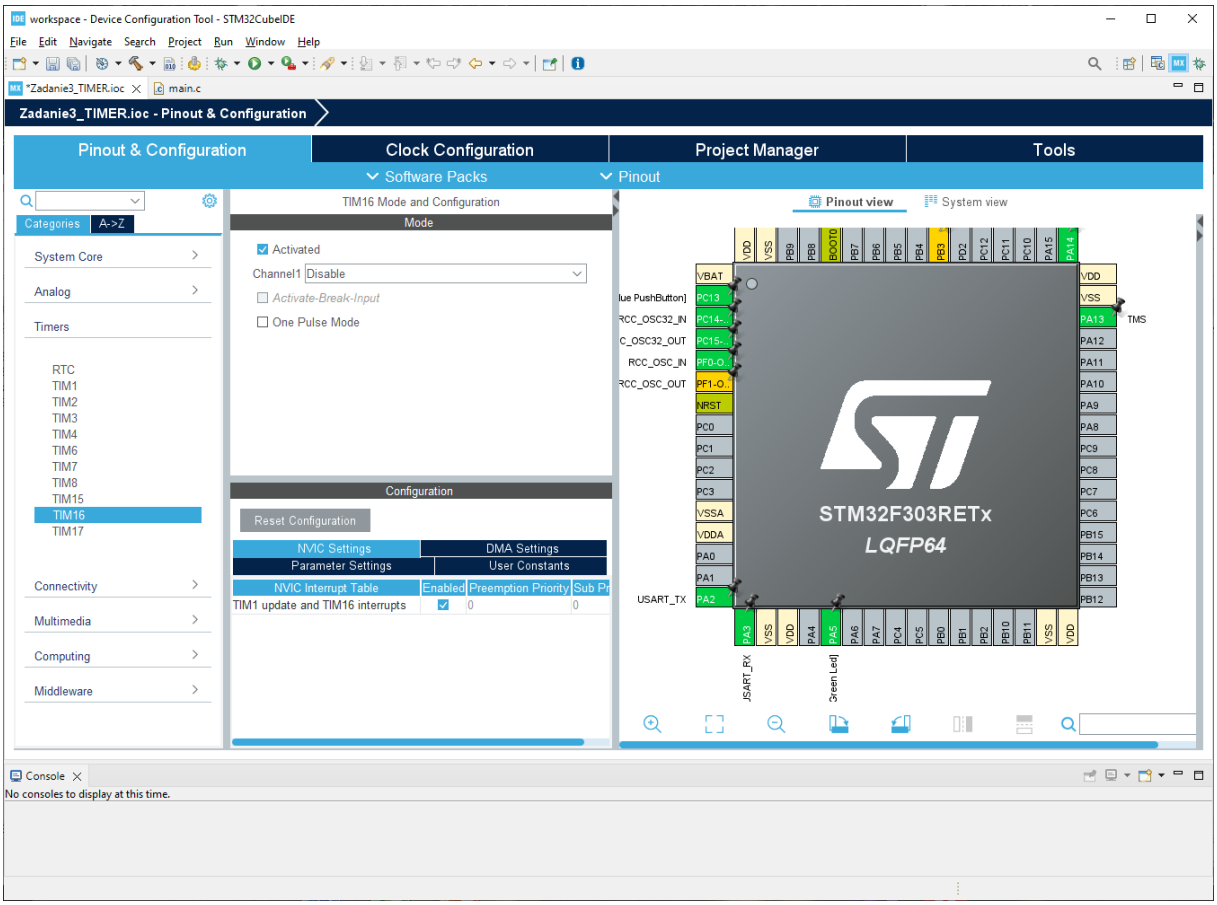
Częstotliwość pojawiających się przerwań można wyznaczyć z poniższego wzoru:

$$FREQ_{int} = \frac{TIMER\_Clk}{(PSC + 1)(CKD + 1)(ARR + 1)}[Hz]$$

gdzie:

$FREQ_{int}$ – częstotliwość generowanych przerwań
$TIMER\_Clk$ – częstotliwość taktowania magistrali, do której dołączony jest TIMER.

Jak widać ze wzoru, rejestry mogą pozostać zerowe. Zatem ustawienia i nastawy rejestrów można ograniczać do niezbędnego minimum. Należy dobrać częstotliwość generacji przerwań zgodnie z poleceniem ustawiając odpowiednie wartości rejestrów **PSC** oraz **ARR**. Rejestr **CKD** nie jest wykorzystywany w bieżącej konfiguracji zatem w ustawieniach i przeliczeniach należy go pominąć, przyjmując **No division** w konfiguracji i zero w powyższym wzorze. Następnie należy aktywować przerwanie od timera – NVIC Setings (Rysunek 3).



Rysunek 3. Aktywacja przerwania TIMERa TIM16 – STM32F303.

Konsekwentnie w funkcji main należy umieścić wywołanie funkcji:

```
HAL_TIM_Base_Start_IT(&htim16);
```

Nowa struktura programu będzie miała postać:

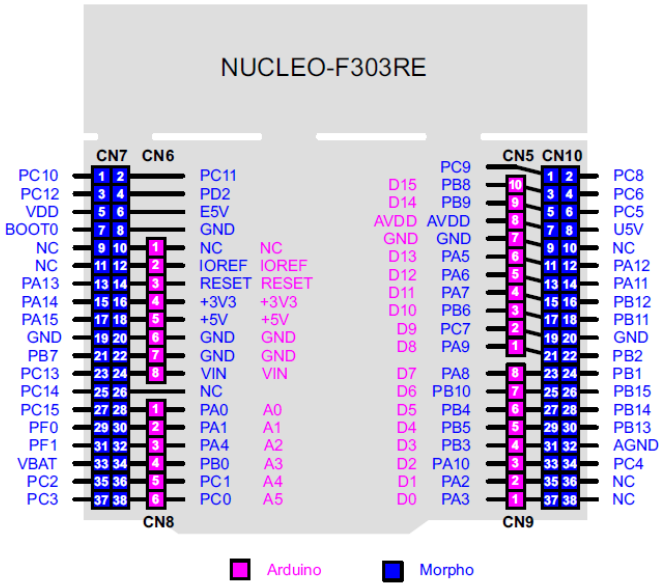
```
TIMER_inicjacja;
TIMER_Start_Przerwania;
while(1)
{
}
ObsługaPrzerwaniaTIMER;
```

Po wprowadzeniu zmian i wygenerowaniu kodu dodana jest funkcja obsługi przerwania. Jest ona umieszczona w pliku źródłowym zawierającym funkcje HAL dla ADC. 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

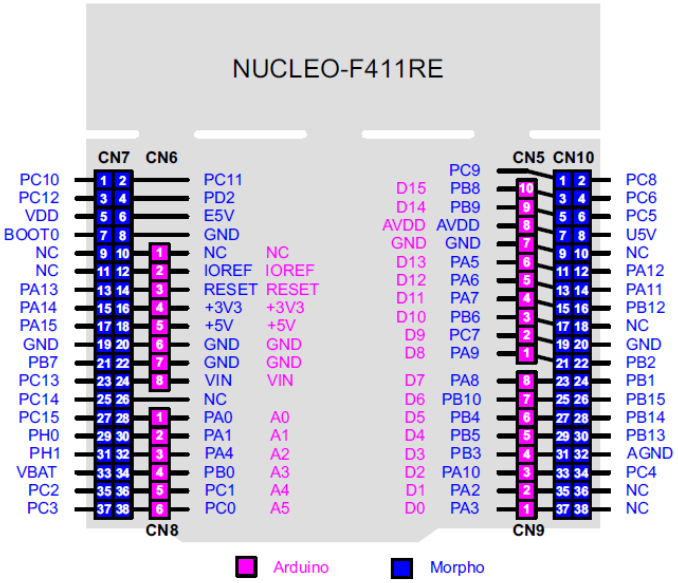
definicji nowej funkcji. W zadaniu nowa funkcja ma nazwę **`HAL_TIM_PeriodElapsedCallback`** oryginalnie/szablonowo znajduje się w pliku stm32fxxx_hal_tim.c. Obsługa przerwania jest nową wygenerowaną funkcją. W niej należy umieścić odczyt stanu przetwornika i skalowanie. Nie trzeba już odpytywać o stan przetwornika i uruchamiać kolejnych konwersji. Przed skompilowaniem i uruchomieniem nowego programu należy dokonać jeszcze jednej zmiany. W programie należy użyć **`HAL_TIM_Base_Start_IT`**, aktywującej mechanizm obsługi przerwań. Po skompilowaniu programu należy go wgrać do mikrokontrolera.

Funkcja przerwania może obsługiwać wiele liczników zatem aby obsłużyć wyłącznie pożądany w funkcji **`HAL_TIM_PeriodElapsedCallback`** warto dodać instrukcję warunkową np.:

```
if(htim->Instance == TIM16)
```



Rysunek 4. Wyprowadzenia na płytce NUCLEO-F303RE. Źródło: STM32 Nucleo-64 boards (MB1136.pdf).



Rysunek 5. Wyprowadzenia na płytce NUCLEO-F411RE. Źródło: STM32 Nucleo-64 boards (MB1136.pdf).

Rysunki 4 i 5 przedstawiają wyprowadzenia dla płytek NUCLEO dla dwóch mikrokontrolerów STM23F411RE i STM23F4303RE. Dla wymienionych mikrokontrolerów przykładowe dostępne wejściowe linie analogowe to: PA0 (IN0), PA1 (IN1) dla Nucleo-STM32F411 albo PC0 (IN6), PC1 (IN7) dla Nucleo-STM32F303.

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.
  *
  ******************************************************************************
  */
```

```c
/* 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 ---------------------------------------------------------*/
 TIM_HandleTypeDef htim16;

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);
static void MX_TIM16_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_USART2_UART_Init();
  MX_TIM16_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};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** 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.PLLMUL = RCC_PLL_MUL9;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  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();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_TIM16;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.Tim16ClockSelection = RCC_TIM16CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{

  /* USER CODE BEGIN TIM16_Init 0 */

  /* USER CODE END TIM16_Init 0 */

  /* USER CODE BEGIN TIM16_Init 1 */

  /* USER CODE END TIM16_Init 1 */
  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 0;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim16.Init.Period = 65535;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM16_Init 2 */

  /* USER CODE END TIM16_Init 2 */

}

/**
```

```
  * @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 = 38400;
  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;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  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_GPIOF_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 TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{
```

```
  /* USER CODE BEGIN TIM16_Init 0 */

  /* USER CODE END TIM16_Init 0 */

  /* USER CODE BEGIN TIM16_Init 1 */

  /* USER CODE END TIM16_Init 1 */
  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 0;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim16.Init.Period = 65535;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM16_Init 2 */

  /* USER CODE END TIM16_Init 2 */

}
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim);

/**
  * @brief  Toggle the specified GPIO pin.
  * @param  GPIOx where x can be (A..F) to select the GPIO peripheral for STM32F3
family
  * @param  GPIO_Pin specifies the pin 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);
}
```