Sterowniki robotów

Laboratorium – Liczniki i przerwania

Tryby pracy licznika oraz obsługa przerwań

Wojciech Domski

Dokument `sr_lab03.pdf`, Wersja 1.0.4

Spis treści

1 Wprowadzenie

Jednym z podstawowych peryferiów występujących w mikrokontrolerach są układy zliczające. Potrafią one generować podstawę czasu o zadanych parametrach, a także pracować w różnych trybach uwzględniających wyjścia, jak i wejścia do peryferium.

2 Opis ćwiczenia

Celem ćwiczenia jest zaznajomienie się z trzema podstawowymi trybami licznika. Mikrokontrolery STM32 z różnych rodzin posiadając szeroką gamę funkcjonalności [7]. Jak zostało wcześniej już nadmienione wlicza się w to m.in.:

- generator podstawy czasu,
- Input Capture,
- Output Capture,
- Pulse Width Modulation (PWM),
- pojedyncze wyzwolenie (One Pulse).

Ponadto, licznik może być taktowany sygnałem zewnętrznym, bądź wewnętrznym. Jedną z dużych zalet liczników w mikrokontrolerach ST jest możliwość łączenia ich w kaskady. Wówczas jeden z liczników jest licznikiem nadrzędnym (ang. *Master*), natomiast drugi podrzędnym (ang. *Slave*). Szczegółowy opis trybów pracy dla mikrokontrole znajdującego się na płytce Nucleo [9] można odnaleźć w [10].

Zakres laboratorium obejmuje zapoznanie się z trzema podstawowymi trybami pracy liczników: generowanie podstawy czasu, Input Capture, a także PWM. Również to ćwiczenie ma na celu przybliżenie działania mechanizmu przerwań, a konkretnie funkcji zwrotnych (ang. *callback*). Szczegółowy opis funkcji przydatnych podczas laboratorium można znaleźć w [6].

Uwaga! Przykłady implementacji różnych peryferiów mikrokontrolera można znaleźć w plikach biblioteki HAL dla danej rodziny układów. Pliki te znajdują się zazwyczaj w ustaloenj ścieżce, np. *C:/Users/Wojciech Domski/STM32Cube/Repository/*. Jednakże położenie tych plików może być różne w zależności od systemu operacyjnego, jak i miejsca instalacji biblioteki. Aby zweryfikować katalog przechowywania repozytorium należy uruchomić program `STM32CubeMX`, a następnie wybrać *Help* → *Updater settings ...* i odczytać zawartość pola *Repository folder*. Przykładami znajdują się w podkatalogu *STM32Cube_FW_L4_V1.7.0/Projects/STM32L476RG-Nucleo/Examples/*. podzielonym na podfoldery ze względu na wykorzystywane peryferium.

3 Zadania do wykonania

Uwaga! Pamiętaj, aby wyłączyć optymalizację kodu (-O0), a także wykorzystać kompilację równoległą (*parallel*) np. -j8 [4]. Ponadto, ustaw automatyczny zapis przed wykonaniem kompilacji, aby uniknąć problemów związanych z rozbieżnością kodu, a plikiem wynikowym wgranym na mikrokontroler. Pamiętaj również, że gdy powtórnie generujesz projekt może okazać się konieczne jego wyczyszczenie jak i przebudowanie indeksu. W tym celu rozwiń menu kontekstowe dla projektu i wybierz *Clean project*, a nastepnie powtórz operację oraz wybierz *Index* → *Rebuild*.

Należy wykonać trzy zadania, które dotyczą generatora podstawy czasu, konfiguracji kanału licznika w trybie *Input Capture* oraz generatora PWM. Do wizualizacji efektów pracy posłuży dioda LED znajdująca się na płytce, a wejście sygnału *Input Capture* będzie symulowane przez niebieski przycisk również znajdujący się na płytce.

3.1 Generator podstawy czasu

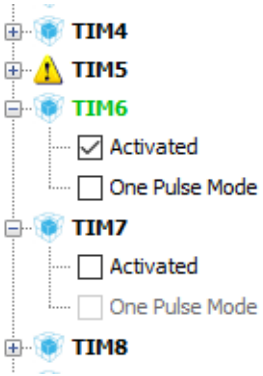
Jednym z podstawowych zastosowań liczników jest ich konfiguracja w trybie generatora podstawy czasu. Do tego celu nie jest wymagane, aby owe peryferium posiadało wejścia, bądź wyjścia. Jednym z częstych zastosowań takiego generatora jest implementacja zegara *SysTick* dla systemu operacyjnego FreeRTOS [1].

Stówrz nowy projekt w programie `STM32CubeMX`. Pamiętaj, aby wybrać płytkę deweloperską NUCLEO-L476RG [8], [9].

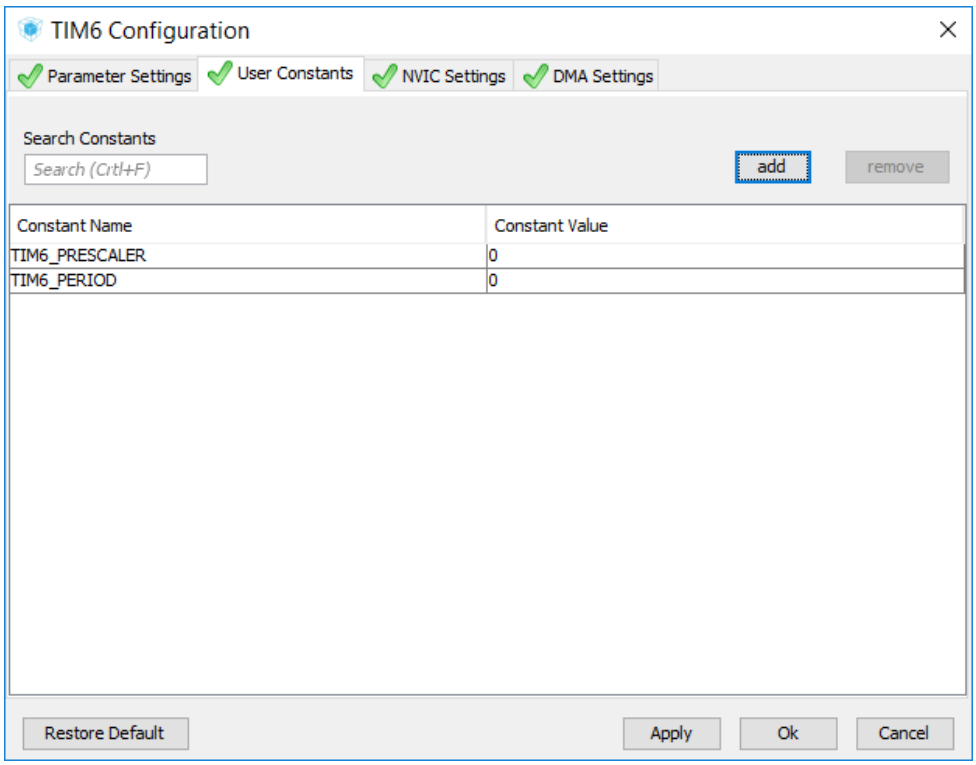
Aktywuj np. licznik *TIM6* tak, jak to zostało pokazane na rysunku 1. Następnie przejdź do konfiguracji licznika. Rozpocznij ją od dodania stałych (2), których wartość nadasz później. Stałe, które należy utworzyć to *TIM6_PRESCALER* oraz *TIM6_PERIOD*. Następnie uruchom globalne przerwanie od licznika 3. Przejdź teraz do zakładki konfiguracji parametrów samego licznika *TIM6* (rys. 4). W miejsce dzielnika sygnału taktującego (ang. *prescaler*) wstaw nazwę wcześniej wprowadzonej stałej *TIM6_PRESCALER*, natomiast do wartości okresu wpisz *TIM6_PERIOD*.

Zastanów się teraz, jakie powinny być wartości stałych *TIM6_PRESCALER* i *TIM6_PERIOD*, jeżeli chcemy, aby przerwanie było generowane co jedną sekundę. Aby wykonać te kalkulacje musisz znać wartość sygnału taktującego licznik *TIM6*. Jest ona pośrednio powiązana z częstotliwością magistrali, do której podłączony jest licznik. Weź również pod uwagę, że licznik jest 16-bitowy, a w tym przypadku, również z nim stowarzyszony rejestr prescalera. Ma to wpływ na maksymalną wartość, jaką te rejestry mogą przechowywać.

Uwaga! Jakie inne tryby pracy posiada licznik TIM6? Do jakiej magistrali podłączony jest licznik *TIM6*, [5], [10]?

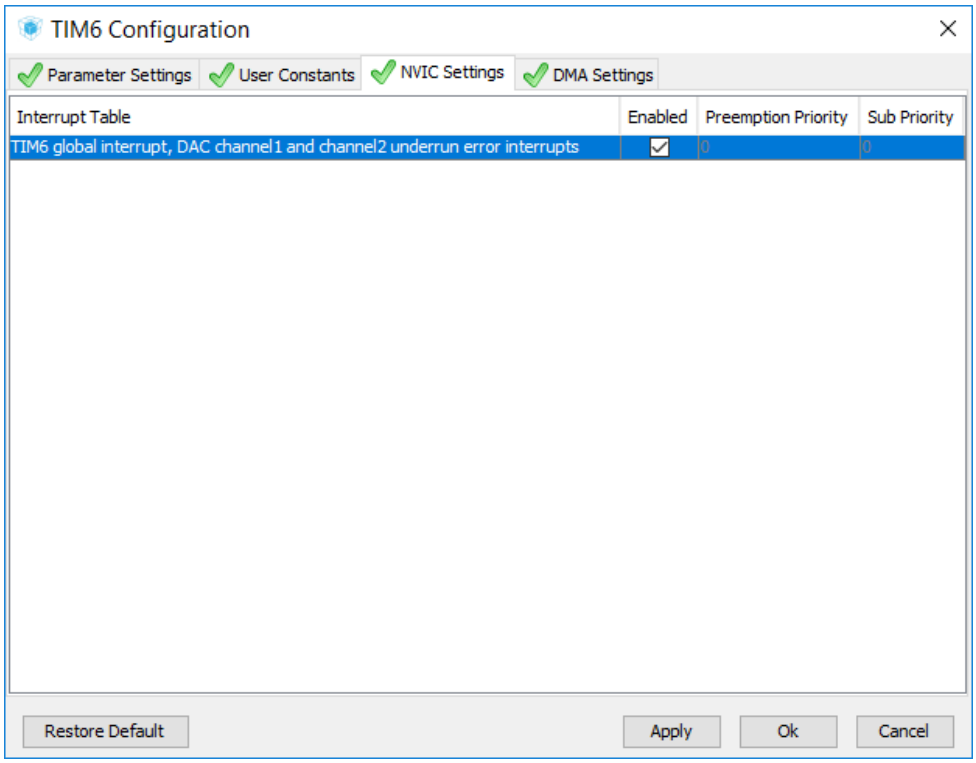


Rysunek 1: Aktywowany licznik TIM6

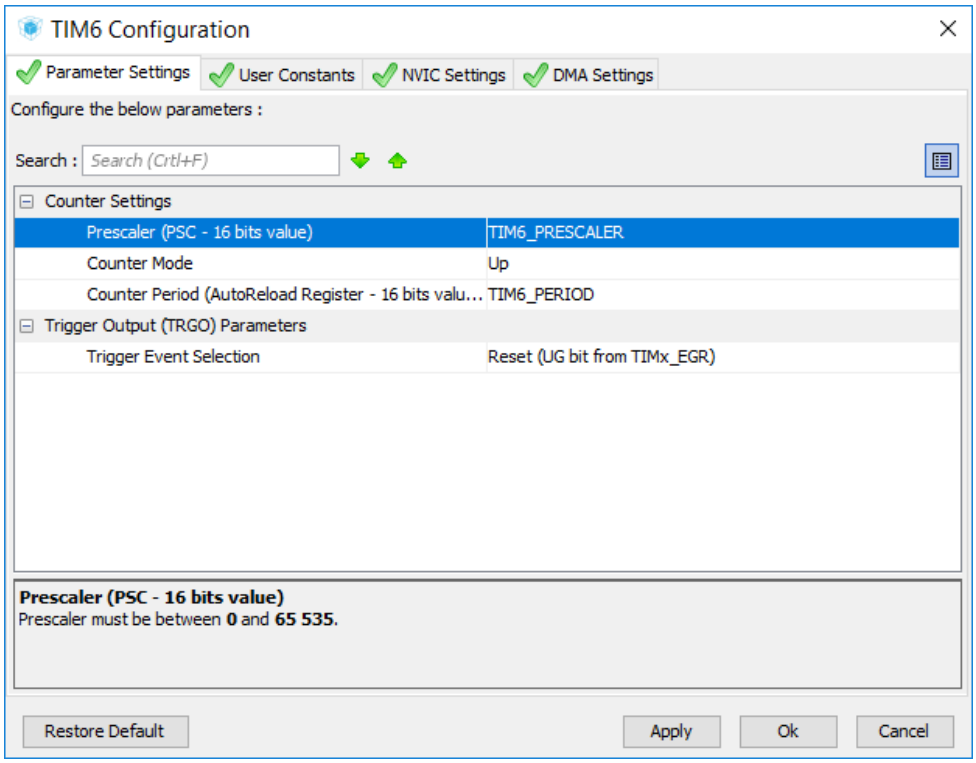


Rysunek 2: Stałe w programie STM32CubeMX

Po ustawieniu odpowiednich wartości stałych wygeneruj kod programu.



Rysunek 3: Włączenie współdzielonego przerwania dla licznika TIM6



Rysunek 4: Ustawienia parametrów licznika TIM6

Uwaga! Dostęp do tych stałych masz również w wygenerowanym kodzie programu. Wszystkie definicje znajdują się w pliku *main.h*. Jednakże podczas ręcznej ich zmiany zostaną one nadpisane po ponownym wygenerowaniu kodu z programu `STM32CubeMX`.

W programie dodaj następującą definicję funkcji:


```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim){
  if(htim == &htim6){
    //obsługa przerwania od licznika
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
}
```

Do czego służy warunek porównujący?

Program wymaga jeszcze jednej modyfikacji. Należy uruchomić licznik. Można to zrobić przez wywołanie odpowiedniej komendy, jak poniżej:

```
  HAL_TIM_Base_Start_IT(&htim6);
```

Skompiluj i uruchom program. Dioda powinna zmieniać swój stan co sekundę.

Uwaga! Jakiego innego licznika można byłoby użyć w zamian za *TIM6*. Czy zamiast funkcji `HAL_TIM_Base_Start_IT` można było by użyć `HAL_TIM_Base_Start`? Jeśli tak, to jak wówczas nazywamy takie podejście? Jeśli nie to dlaczego?

3.1.1 Przerwania

Jednym z podstawowych mechanizmów występujących w mikrokontrolerach są przerwania. Przerwania mogą być wewnętrzne, czyli takie, które generowane są przez peryferia mikrokontrolera, bądź zewnętrzne. Przerwania zewnętrzne stowarzyszone są z zewnętrznym źródłem wyzwalającym (ang. *external trigger source*), które pośrednio lub bezpośrednio podłączone jest do pinu układu.

Przykładem wewnętrznego przerwania jest przepełnienie licznika. Wykorzystywane jest ono w tym ćwiczeniu, ale tylko pośrednio. Wraz z wprowadzeniem biblioteki *Hardware Abstraction Layer* (HAL) [6] bezpośredni dostęp do obsługi przerwań został ograniczony. W zamian za to wprowadzono tzw. wywołania zwrotne (ang. *callback*). Korzystanie z tego typu obsługi przerwań zostało uproszczone do tego stopnia, że w programie wystarczy reimplementować odpowiednią funkcję zwrotną.

Jak działa obsługa przerwań? Każde przerwanie (jego adres) jest wpisane do tzw. tablicy przerwań. Znajduje się ona zazwyczaj w pliku startowym **.s*, którego fragment został zamieszczony poniżej:

```
g_pfnVectors:
.word  _estack
.word  Reset_Handler
.word  NMI_Handler
.word  HardFault_Handler
.word  MemManage_Handler
.word  BusFault_Handler
.word  UsageFault_Handler
.word  0
.word  0
.word  0
.word  0
.word  SVC_Handler
.word  DebugMon_Handler
.word  0
.word  PendSV_Handler
.word  SysTick_Handler
.word  WWDG_IRQHandler
.word  PVD_PVM_IRQHandler
//  ...
.word  TIM6_DAC_IRQHandler
//  ...
```

Przykładowo weźmy pod uwagę przerwanie `TIM6_DAC_IRQHandler` wykorzystywane w ćwiczeniu. Jego implementacja jest następująca i znajduje się w pliku *stm32l4xx_it.c*.

```
/**
* @brief This function handles TIM6 global interrupt, DAC channel1 and channel2 underrun error
    interrupts.
*/
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Jak widać przerwanie dla licznika TIM6 (w tym mikrokontrolerze) jest współdzielone z przerwaniem od przetwornika DAC co sugeruje nazwa.

Jak już zostało to nadmienione, tablica przerwań znajduje się w specjalnym obszarze pamięci pod adresem 0x00000000 [10]. Należy jednak zauważyć, że obsługa przerwania w przypadku biblioteki HAL

jest dwustopniowa, a nie jedno, jak to było dla biblioteki SPL (Standard Peripheral Library) lub innych. Podejście to ma zalety, jak i wady. Niezaprzeczalną zaletą jest prostota obsługi przerwań, wadą natomiast kolejny poziom zagłębienia.

Kiedy przychodzi przerwanie od danego peryferium mikrokontroler automatycznie próbuje wywołać funkcję spod danego adresu w tablicy wektorów przerwań *g_pfnVectors*. Następnie wywoływana jest procedura obsługi przerwania np. przytoczona wcześniej funkcja `TIM6_DAC_IRQHandler`, gdzie wywoływana jest kolejna funkcja `HAL_TIM_IRQHandler`, w której to obsługiwane są flagi przerwania, a następnie wywoływana jest funkcja zwrotna (ang. callback). Przykład implementacji funkcji `HAL_TIM_IRQHandler` został zaprezentowany poniżej.

```
void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim)
{
  //...
  /* TIM Update event */
  if(__HAL_TIM_GET_FLAG(htim, TIM_FLAG_UPDATE) != RESET)
  {
    if(__HAL_TIM_GET_IT_SOURCE(htim, TIM_IT_UPDATE) !=RESET)
    {
      __HAL_TIM_CLEAR_IT(htim, TIM_IT_UPDATE);
      HAL_TIM_PeriodElapsedCallback(htim);
    }
  }
  /* TIM Break input event */
  //...
}
```

Sama funkcja zwrotna posiada następującą (słabą) definicję:

```
__weak void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is needed,
  the __HAL_TIM_PeriodElapsedCallback could be implemented in the user file
  */
}
```

Dzięki zdefiniowaniu funkcji zwrotnej jako *__weak* możliwe jest jej nadpisanie przez linkera na etapie konsolidacji.

Najważniejszą zasadą dotyczącą obsługi przerwania jest jego czas. Należy tak pisać program, aby przebywanie w przerwaniu trwało możliwie, jak najkrócej. Jedną z metod jest ustawianie flag w programie, które są cyklicznie sprawdzane w głównym wątku lub zadaniach (w przypadku systemu czasu rzeczywistego FreeRTOS [2]). W obsłudze przerwania w żadnym wypadku nie wolno wykonywać czasochłonnych obliczeń np. macierzowych.

3.2 Input Capture

Input Capture, jako tryb pracy licznika jest bardzo przydatny, gdy chcemy wykonać pomiar czasu pomiędzy zdarzeniami. Wyobraź sobie, że uczestniczysz w zawodach linefollowerów. Jak zmierzyć czas przejazdu robota? Można ustawić jedną bramkę z fotodiodą na torze, po którym porusza się robot. W momencie, gdy przejeżdża pod bramką pojazd generowany jest sygnał elektryczny od czujnika (np. fotodiody) o odpowiednim zboczu (narastającym, bądź opadającym). Sygnał ten jest doprowadzany do pinu mikrokontrolera, który skonfigurowany jest, jako wejście typu Input Capture. W chwili, w której zostanie zarejestrowane zbocze na wejściu układu zatrzaskiwana jest aktualna wartość rejestru licznika, a sam rejestr może być wyczyszczony. Znając częstotliwość z jaką pracuje licznik oraz zatrzaśniętą wartość można w prosty sposób wyliczyć czas od ostatniego zdarzenia.

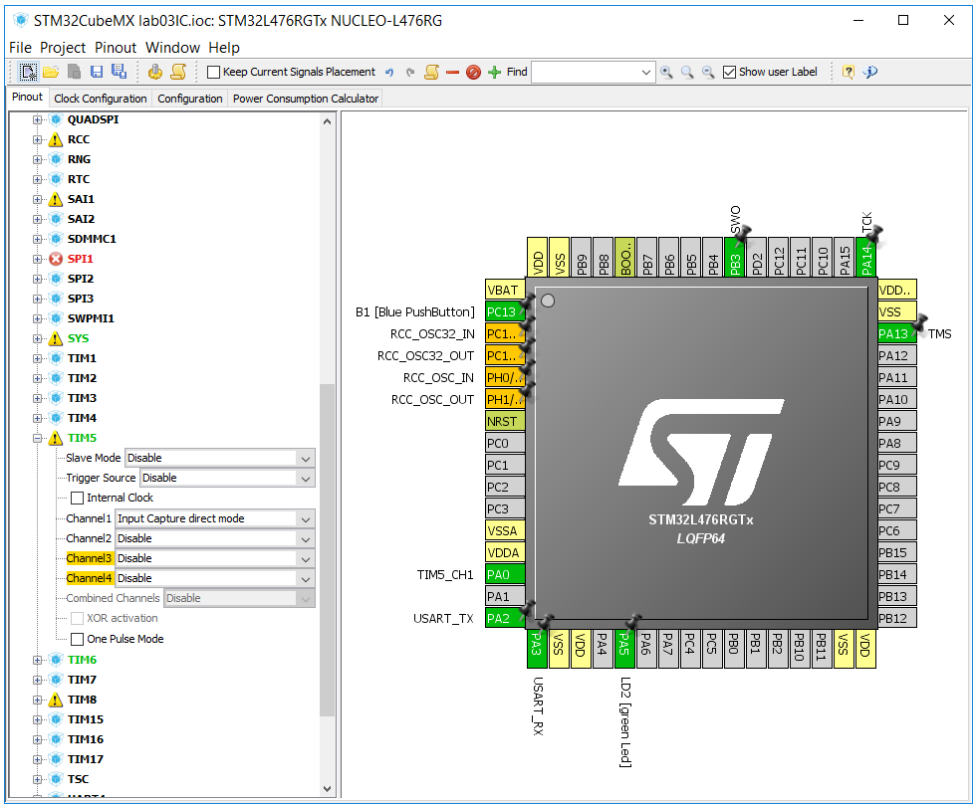
Stwórz nowy projekt w programie `STMCube32Mx` na podstawie szablonu do płytki deweloperskiej NUCLEO-L476RG. W tym ćwiczeniu zostanie wykorzystane również przekierowanie funkcji `printf()` [3].

Skonfiguruj licznik TIM5, tak jak pokazano na rysunku 5. Wówczas pin PA0 powinien automatycznie podświetlić się na zielono.

Przejdź do zakładki *Configuration* i rozpocznij konfigurację peryferium TIM5. Rozpocznij od dodania stałych *TIM5_PRESCALER* i *TIM5_PERIOD* w zakładce *User constants*, jak pokazano na rysunku 6.

Powróć do zakładki *Parameter Settings* i ustaw kolejno wartości dla *Prescaler (PSC - 16 bits value)* oraz *Counter period (AutoReload Register - 32 bits value)*. Ustaw również zbocze na opadające (ang. *Falling Edge*) w *Input Capture Channel 1* → *Polarity Selection*. Poprawna konfiguracja została pokazana na rysunku 7.

Następnie przejdź do zakładki przerwań *NVIC Settings* oraz uaktywnij przerwanie (rys. 8). Na ostatniej zakładce – wejść/wyjść *GPIO Settings* wybierz rekord z nazwą pinu PA0, a następnie z menu, które



Rysunek 5: Aktywacja licznika TIM5 oraz wejścia Input Capture

się pojawi ustaw podciągnięcie portu do napięcia zasilania (ang. *Pull-up*), jak zostało to przedstawione na rysunku 9.

Uwaga! Ilo bitowym jest licznik TIM5. Czy możesz wymienić inny licznik o takiej samej liczbie bitów?

Ponadto, skonfiguruj port szeregowy USART2 w asynchronicznym trybie pracy, tak, jak to pokazano w [4]. Wygeneruj kod z programu `STM32CubeMx` oraz przystąp do jego modyfikacji.

Dodaj następujące definicje zmiennych globalnych do programu:

```
volatile int flag;
volatile int period;
```

Przed główną pętlą `while()` w programie dodaj następujący kod:

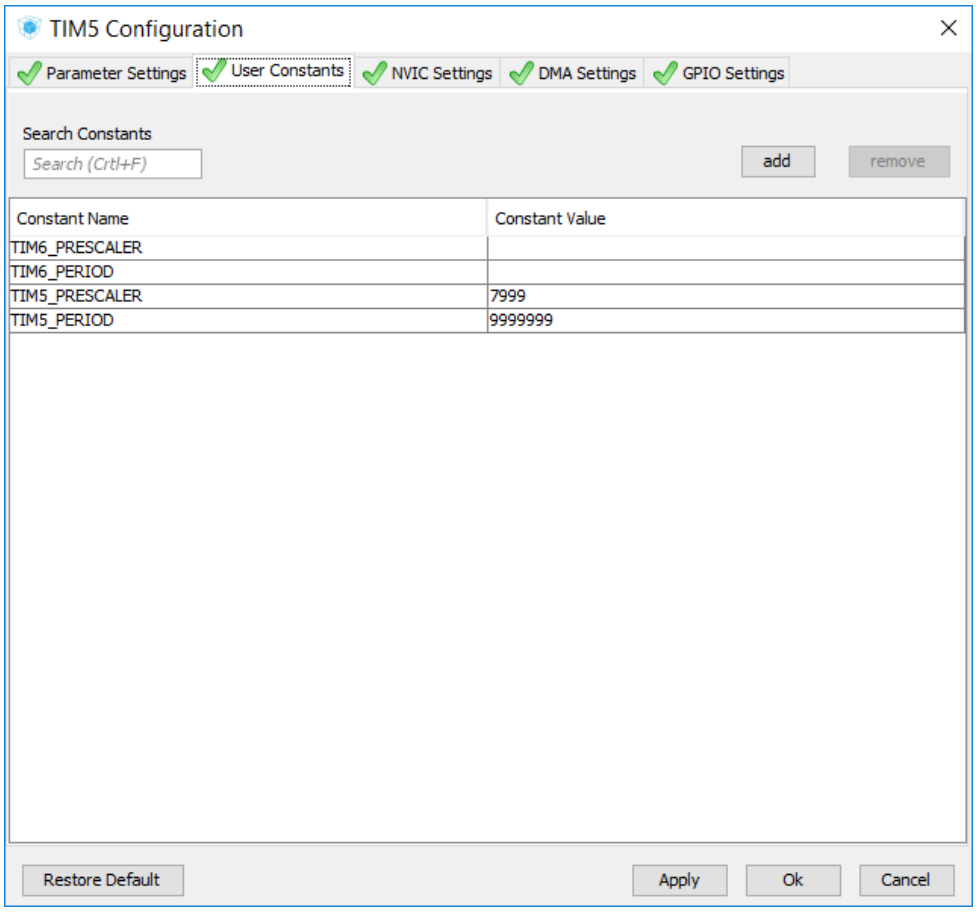
```
flag = 0;
HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_1);
```

Następnie uzupełnij pętlę `while()` następującymi instrukcjami:

```
if (flag == 1) {
  HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  printf("Wykryto zbocze po %d sekundach\r\n", period/10000);

  flag = 0;
}

HAL_Delay(10);
```

Zdefiniuj następujące dwie funkcje w programie:

```
int __write(int file, char *ptr, int len) {
  HAL_UART_Transmit(&huart2, ptr, len, 50);
  return len;
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef * htim) {
  if (htim == &htim5) {
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
      period = __HAL_TIM_GET_COMPARE(&htim5, TIM_CHANNEL_1);
      __HAL_TIM_SET_COUNTER(&htim5, 0);
      flag = 1;
    }
  }
}
```

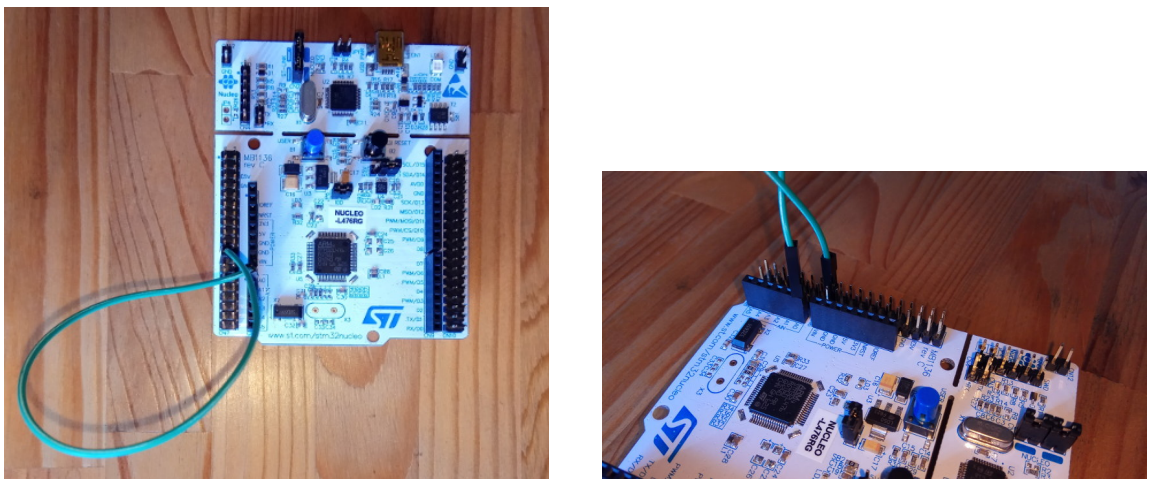


Rysunek 6: Stałe dla licznika TIM5

Uwaga! Pamiętaj również o dodaniu nagłówka *stdio.h*.
Wykonaj kolejno następujące czynności, aby przećwiczyć działanie licznika w trybie Input Capture:

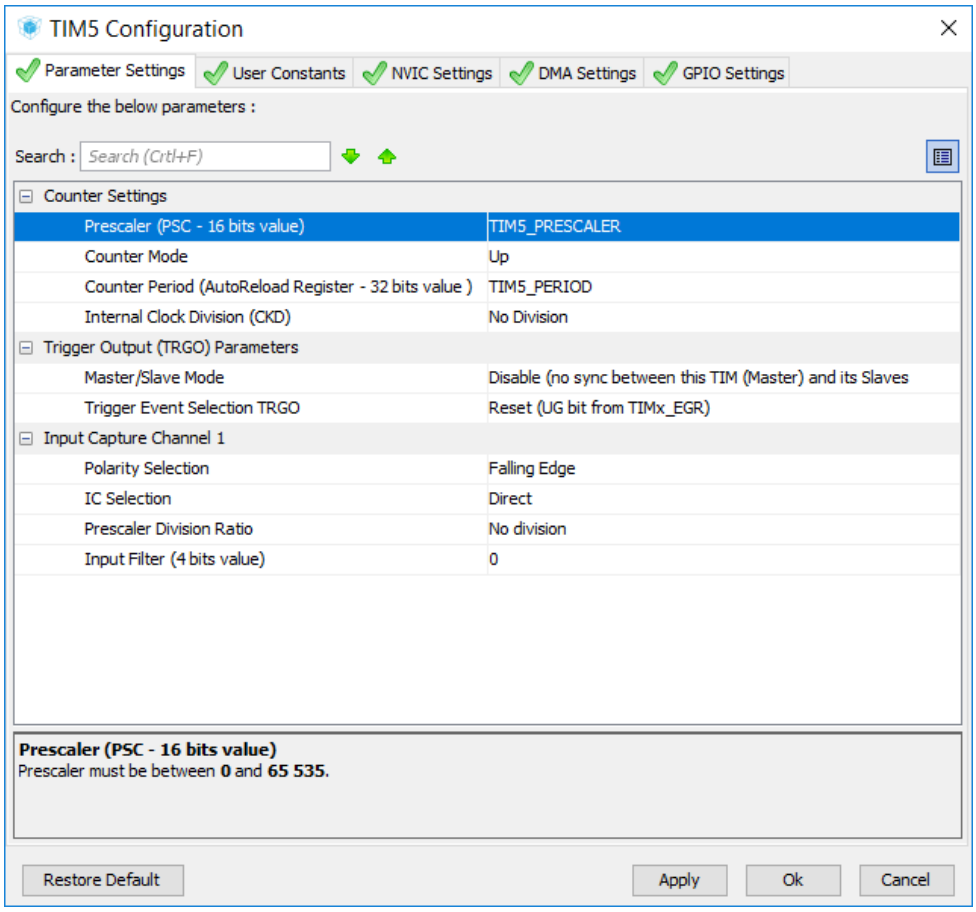
1. Odłącz płytkę od komputera.
2. Połącz przewodem piny PC13 (końcówka żeńska) oraz PA0 (końcówka męska lub żeńska).

Prawidłowo podłączony przewód do płytki został przedstawiony na zdjęciach 10(a) i 10(b).



(a) (b)

Rysunek 10: Zdjęcia przedstawiające prawidłowo podłączony przewód do płytki



Rysunek 7: Parametry licznika TIM5

Uwaga! Zasygnalizuj prowadzącemu wykonanie zadania do tego etapu. Po zgodzie prowadzącego można kontynuować ćwiczenie. Niezgłoszenie prowadzącemu wykonania ćwiczenia do tego etapu wiązać będzie się z konsekwencjami.

Podłącz płytkę do komputera, a następnie uruchom program `Termite` oraz wybierz odpowiedni port. Naciskaj niebieski przycisk w kilku sekundowych i obserwuj pojawiające się komunikaty na porcie szeregowym.

Zadanie dodatkowe Skonfiguruj tak licznik TIM5, aby następowało sprzętowe resetowanie zawartości rejestru licznika (ustawianie wartości 0). Pamiętaj o usunięciu linii

```
1 __HAL_TIM_SET_COUNTER(&htim5, 0);
```

z obsługi przerwania.

Uwaga! Jaki ma sens stała *TIM5_PRESCALER*? Dlaczego nie jest ona ustawiona na wartość 8000. Czy wartość *TIM5_PERIOD* mogła by być inna, jakie miało by to konsekwencje?

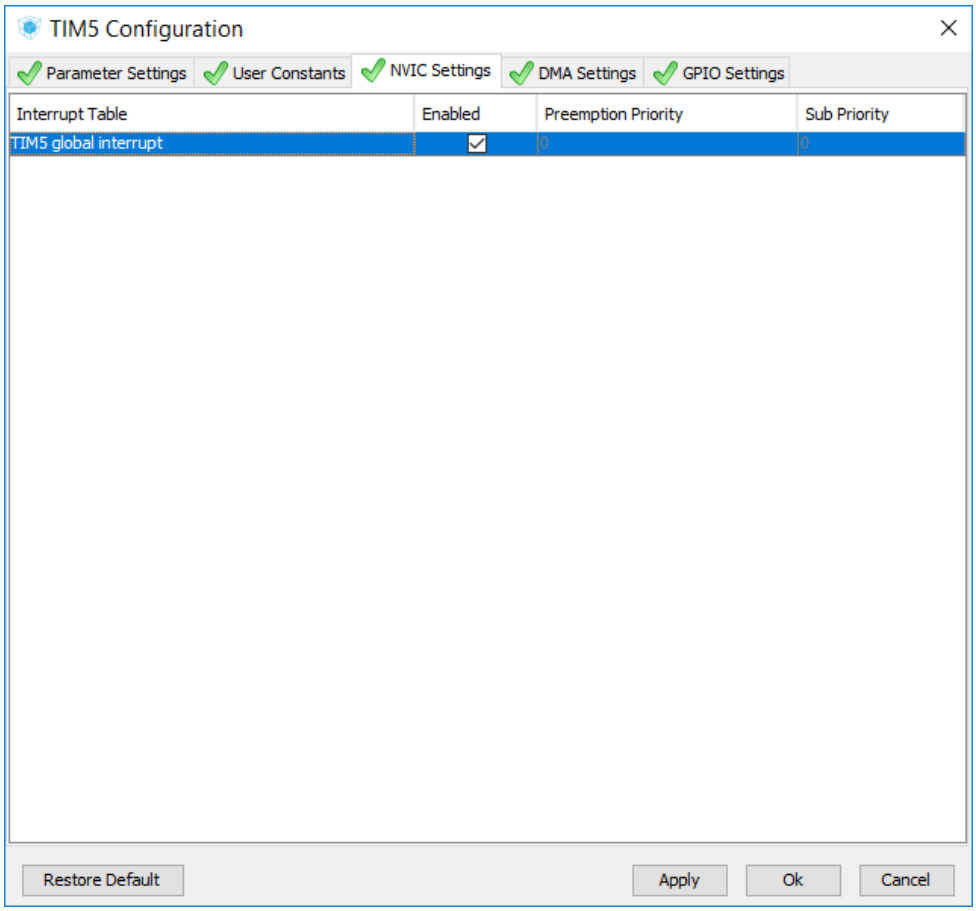
Uwaga! Jak zachowywałby się program, gdyby kanał Input Capture był ustawiony na zbocze narastające lub zarówno na zbocze narastające, jak i opadające?

Uwaga! Po realizacji tego ćwiczenia odłącz płytkę od komputera, a następnie odepnij przewód połączeniowy.

3.3 Pulse Width Modulation

Celem tego ćwiczenia jest zapoznanie się z modulacją szerokości impulsu (PWM) oraz przerwaniami zewnętrznymi (ang. *External Interrupt* EXTI). PWM jest najczęściej wykorzystywany np. do sterowania prędkością obrotową silników. Sam sygnał ma dwa podstawowe parametry,

1. Częstotliwość, bądź okres.
2. Wypełnienie (najczęściej podawane jako wartość procentowa).



Rysunek 8: Konfiguracja przerwań dla licznika TIM5

Utwórz nowy projekt w programie `STM32CubeMx` dla płytki NUCLEO-L476RG. Do wizualizacji procentu wypełnienia zostanie wykorzystana dioda LED przekładając się na jasność świecenia diody.

Skonfiguruj pin PA5, jako generator PWMa. Zostało to zaprezentowane na rysunku 11. W tym celu kliknij prawym przyciskiem myszy na pin PA5, a następnie wybierz *TIM2_ CH1*. W kolejnym kroku dla kanału pierwszego *TIM2* → *Channel1* wybierz *PWM Generator CH1*.

Przejdź do konfiguracji *Configuration* i ustaw parametry licznika TIM2. Rozpocznij od dodania stałych użytkownika (rys. 12). Na końcu skonfiguruj parametry, jak to pokazano na rysunku 13.

Po skonfigurowaniu licznika TIM2 przejdź do konfiguracji zewnętrznego przerwania EXTI. Proces ten został pokazany na rysunku 14. Należy zaznaczyć rekord *EXTI line [15:10] interrupt*. [15:10] świadczy o tym, że tylko linie z numerami od 10 do 15 są obsługiwanie. Dla pozostałych linii wykorzystywane jest inne przerwanie. W przypadku, gdy nie jest on wyświetlony upewnij się, że opcja wyświetlania tylko włączonych przerwań *Show only enabled interrupts* jest odznaczona.

Wygeneruj kod oraz zaimportuj projekt do AC6. Przystąp do modyfikacji kodu. W pliku *main.c* dodaj definicje zmiennych globalnych:

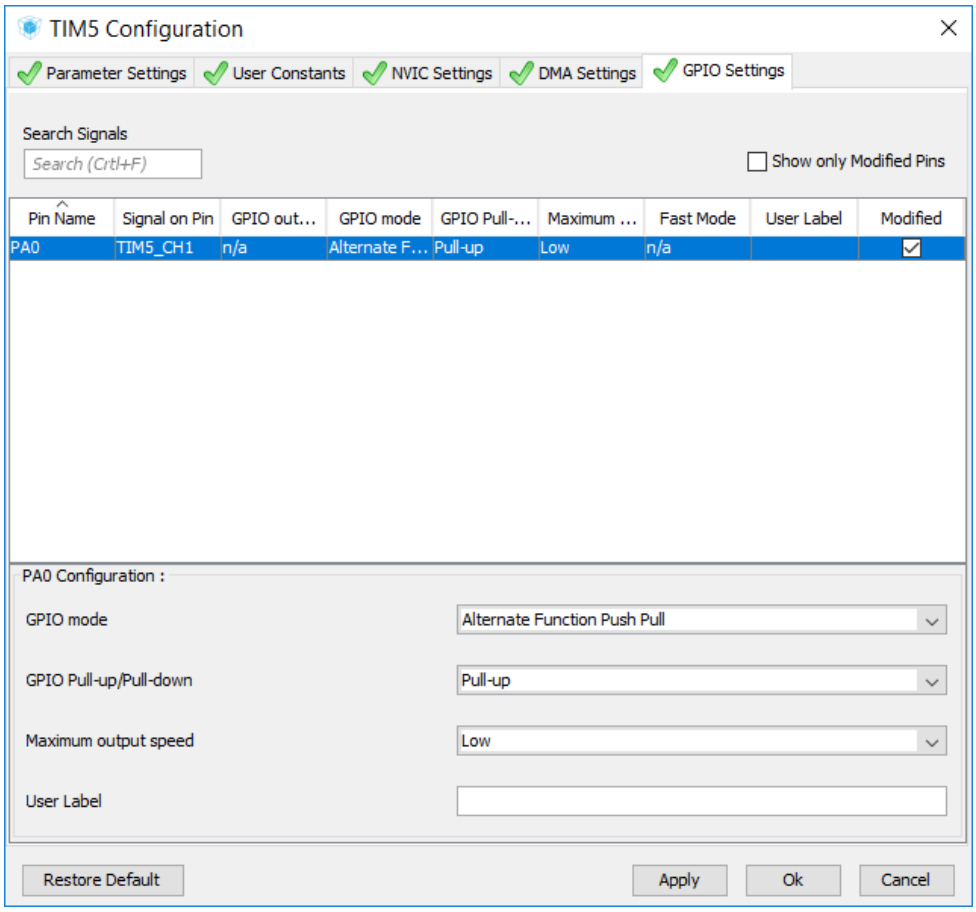
```
int pwm_duty;
volatile int flag;
```

Przed wejściem do pętli `while()` dodaj poniższy kod:

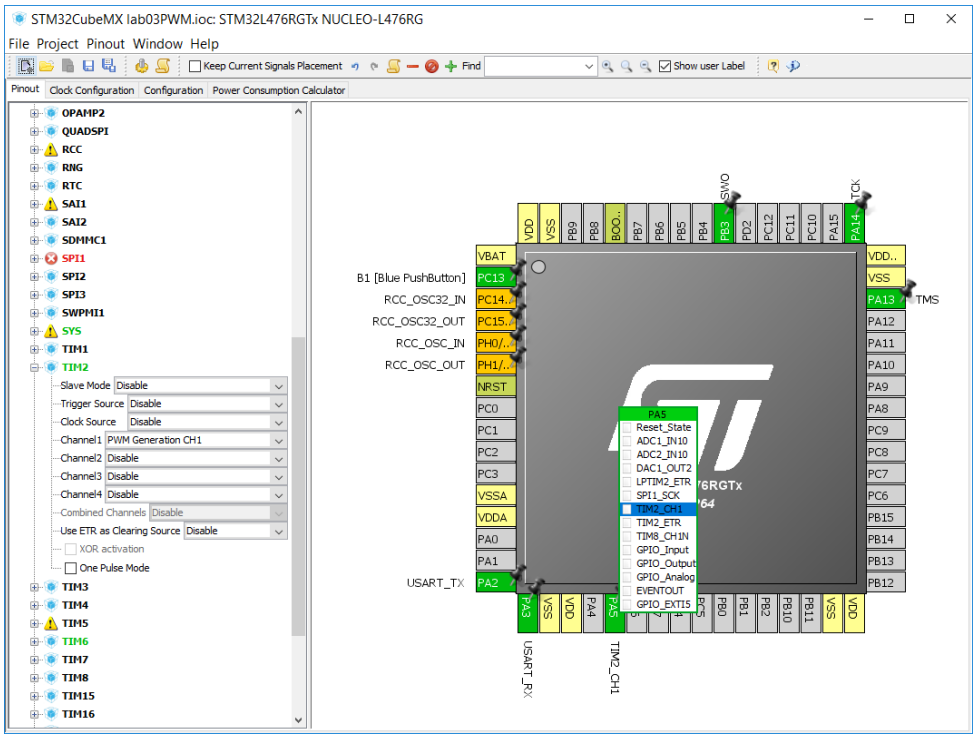
```
flag = 0;
pwm_duty = 0;
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

Natomiast ciało pętli `while()` uzupełnij o poniższe instrukcje:

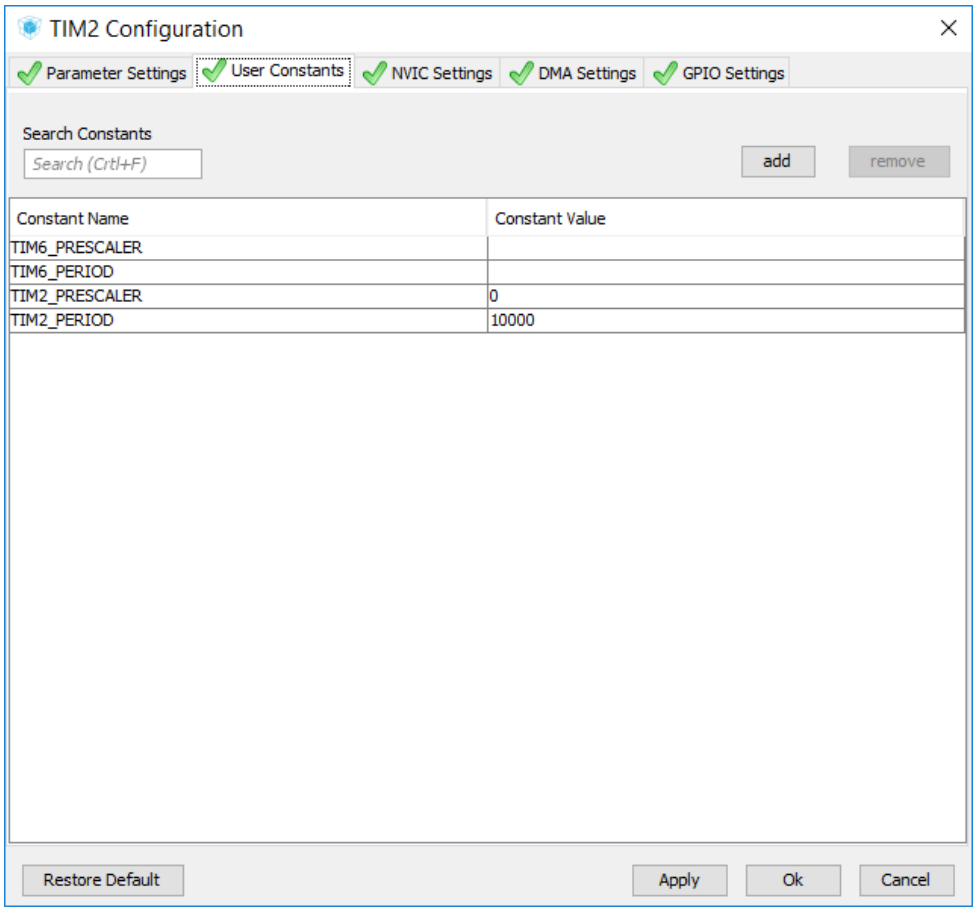
```
if (flag == 1) {
  pwm_duty = (pwm_duty + 1000) % 10000;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pwm_duty);
  flag = 0;
}
HAL_Delay(10);
```

Rysunek 9: Konfiguracja wejść/wyjść przypisanych do licznika TIM5



Rysunek 11: Konfiguracja kanału pierwszego dla licznika TIM2



Rysunek 12: Konfiguracja stałych dla licznika TIM2

Również zdefiniuj funkcję zwrotną dla przerwania zewnętrznego EXTI.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if (GPIO_Pin == GPIO_PIN_13) {
    //obsługa przerwania od przycisku
    flag = 1;
  }
}
```

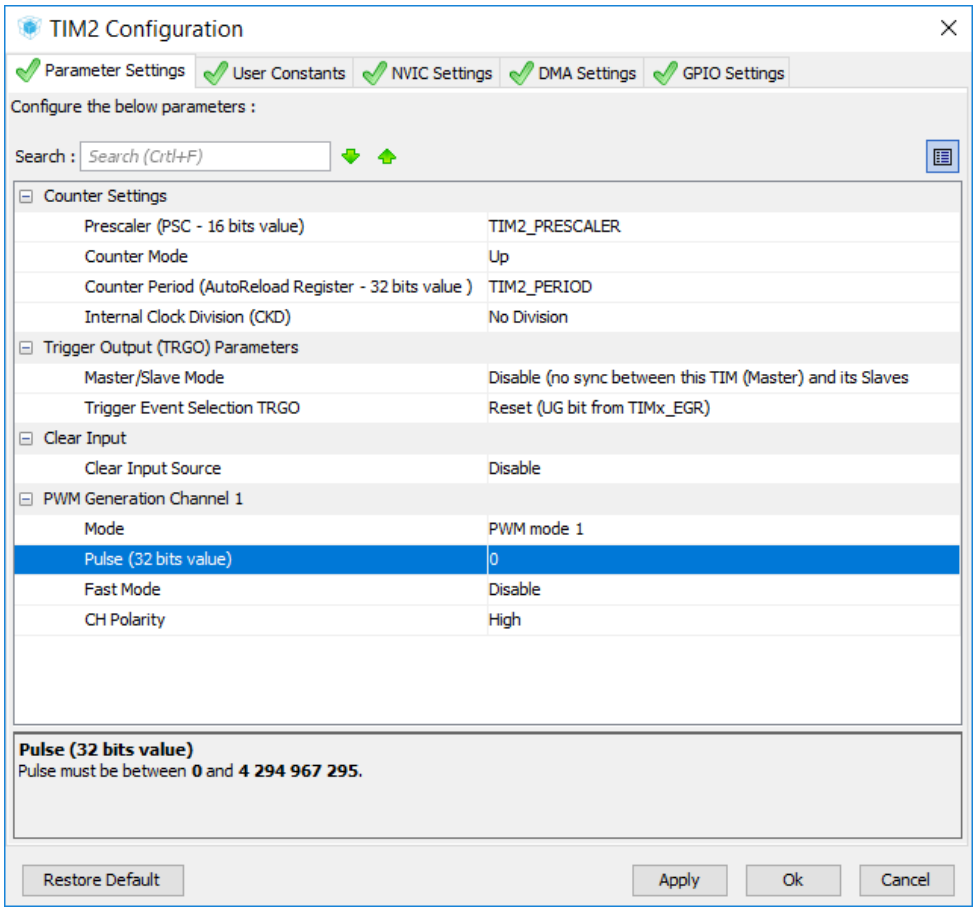
Uwaga! Czy `HAL_GPIO_EXTI_Callback` jest ograniczona tylko do obsługi linii [15:10]? Jak ogólnie działa program? Jaka jest częstotliwość sygnału PWM, który steruje diodą dla przyjętych wartości *TIM2_PRESCALER*. Czy stała *TIM6_PERIOD* ma jakikolwiek wpływ na jakość generowanego sygnału PWM?

3.4 Uporządkowanie stanowiska

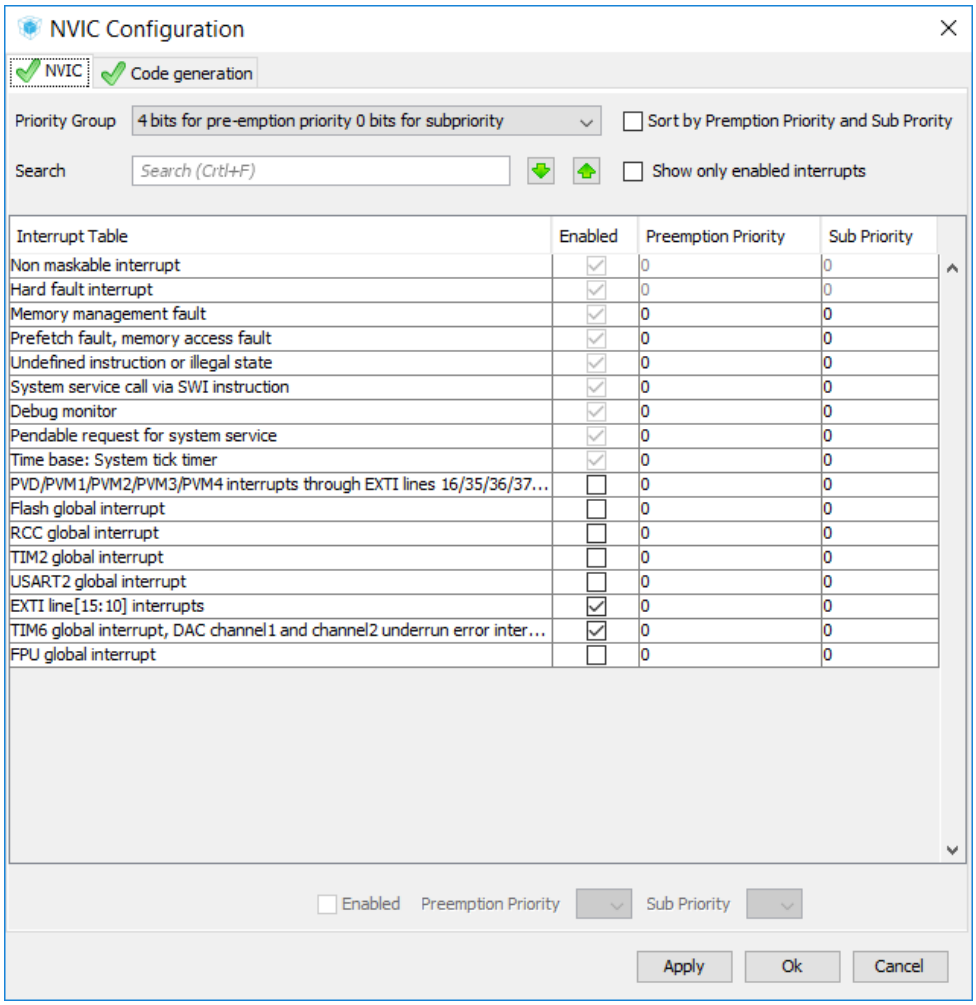
Odłóż płytkę i kabel na miejsce. Usuń projekt z AC6. Można to zrobić przez kliknięcie prawym przyciskiem myszki na projekt i wybranie opcji Usuń (Delete) z menu kontekstowego.

4 Podsumowanie

Instrukcja ta przedstawia w zwięzły sposób obsługe podstawowych trybów pracy liczników w mikrokotrolerach STM32, a także obsługę przerwań wewnętrznych, jak i zewnętrznych. Szczególną uwagę poświęcono trzem trybom pracy licznika: generator podstawy czasu, generator sygnału PWM, a także wejście typu Input Capture.



Rysunek 13: Konfiguracja parametrów licznika TIM2



Rysunek 14: Konfiguracja przerwania EXTI

Literatura

[1] *The FreeRTOS™, Reference Manual*. wydanie 2, Real Time Engineers Ltd.

[2] R. Barry. *Mastering the FreeRTOS™ Real Time Kernel – A Hands-On Tutorial Guide*. wydanie pre, Real Time Engineers Ltd.

[3] W. Domski. Sterowniki robotów, Laboratorium – Debugowanie, Zaawansowane techniki debugowania. Marzec, 2017.

[4] W. Domski. Sterowniki robotów, Laboratorium – Wprowadzenie, Wykorzystanie narzędzi STM32CubeMX oraz SW4STM32 do budowy programu mrugającej diody z obsługą przycisku. Marzec, 2017.

[5] ST. *STM32L476xx, Ultra-low-power ARM® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, analog, audio.*, Grudzień, 2015.

[6] ST. *Description of STM32L4 HAL and Low-layer drivers.*, Luty, 2016.

[7] ST. *STM32 cross-series timer overview, Application note.*, Lipiec, 2016.

[8] ST. *STM32 Nucleo-64 board.*, Listopad, 2016.

[9] ST. *STM32 Nucleo-64 board, User manual.*, Listopad, 2016.

[10] ST. *STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs, Reference Manual.*, Marzec, 2017.

