Systemy mikroprocesorowe

Instrukcja laboratoryjna

Spotkanie 4

Liczniki i konwertery ADC

Autorzy: dr inż. Paweł Dąbal

Ostatnia aktualizacja: 25.11.2022 r.

Wersja: 2.0.0

Warszawa 2022

Spis treści

1. Wprowadzenie

Instrukcja ta zawiera zadania związane z tworzeniem programów obsługujących liczniki i przetworniki ADC dostępne w mikrokontrolerze. W trakcie zajęć student będzie korzystał z środowiska programistycznego *STM32CubeIDE*, rozbuduje dostarczony projekt aplikacji dla mikrokontrolera STM32L496ZGT6 umieszczonego na płycie uruchomieniowej *KAmeLeon* o funkcjonalność umożliwiającą obsłużenie liczników sprzętowych i konwerterów ADC. Ponadto będzie potrafił obsługiwać narzędzia wspomagające kontrolę wersji oprogramowania w celu dokumentowania własnych postępów.

1.1. Cel ćwiczenia

Ćwiczenie laboratoryjne ma na celu:

- nabycie umiejętności konfiguracji stanowiska pracy w oparciu o oprogramowanie *STM32CubeIDE 1.7.0*;
- poznanie płyty uruchomieniowej *KAmeLeon* z mikrokontrolerem *STM32L496ZGT6*;
- przypomnienie i usystematyzowanie wiedzy i umiejętności z zakresu posługiwania się językiem *C*;
- przypomnienie wiedzy z zakresu przetworników ADC;
- przypomnienie wiedzy z zakresu liczników dostępnych w mikrokontrolerach (budowa, zasada działania);
- praktyczne korzystanie z systemu kontroli wersji *Git* i serwisu *GitHub*.

1.2. Wymagania wstępne

Przed przystąpieniem do wykonywania ćwiczenia laboratoryjnego należy:

- zapoznać się z schematem płyty uruchomieniowej *KAmeLeon* oraz dokumentacją mikrokontrolera *STM32L496ZGT6* (*Product Specifications*, *Reference Manuals*);
- zapoznać się z składnią języka C – pojęcie zmiennej, stałej, funkcji, prototypu funkcji, parametru funkcji, wyrażenia warunkowego, pętli, wskaźnik, struktury i tablicy;
- zapoznanie się z sposobem działania liczników oraz konwerterów ADC;
- utworzyć konto na platformie *GitHub*;
- zapoznać się z dokumentacją środowiska *STM32CubeIDE* i biblioteki *STM32CubeL4*.

1.3. Opis stanowiska laboratoryjnego

Stanowisko do przeprowadzenia zajęć składa się z komputera PC z zainstalowanym oprogramowaniem koniecznym do realizacji zajęć: *STM32CubeIDE*, dedykowaną do układu biblioteką *STM32CubeL4* oraz płyty *KAmeleon* podłączonej do komputera za pomocą przewodu USB do złącza programatora *SWD PRG/DBG/vCOM* płyty uruchomieniowej. Połączenie to również odpowiada za zasilanie płyty.

1.4. Wprowadzenie teoretyczne

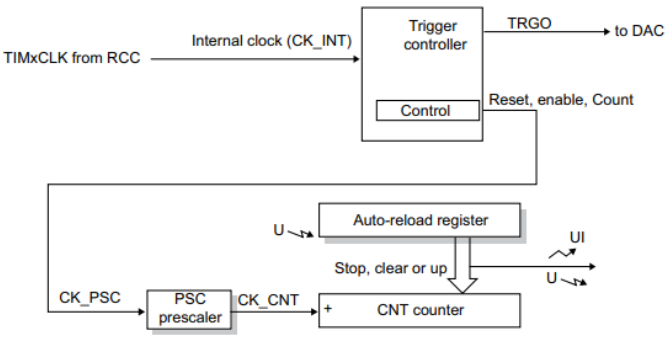
Współczesne mikrokontrolery wyposażone są w wiele różnych modułów peryferyjnych. Jedną z podstawowych grup są sprzętowe liczniki, które mogą się różnić między sobą sposobem implementacji udostępniającym różne tryby pracy. Druga grupa modułów peryferyjnych to przetworniki analogowo-cyfrowe (ADC) i cyfrowo-analogowe (DAC) umożliwiających współpracę z układami analogowymi w ramach urządzenia.

1.4.1. Liczniki sprzętowe[1]

Odliczanie czasu, generowanie sygnału PWM i pomiar czasu trwania impulsu to tylko przykłady zadań do których używane są liczniki (ang. *timer*). W dużym skrócie można powiedzieć, że liczniki to moduły wewnątrz mikrokontrolerów, które zliczają „jakieś" impulsy i sygnalizują fakt doliczenia do ustalonej przez nas wartości. Mogą one zliczać takty sygnału zegarowego lub np. występowanie konkretnego zbocza na wejściu układu. W wyniku działania liczników mogą być generowane przerwania, ale mogą też dziać się inne rzeczy. Co ważne, te wszystkie operacje realizowane są sprzętowo, więc dzieją się niejako w tle i nie obciążają układu. Mikrokontrolery STM32 znane są m.in. właśnie z rozbudowanych liczników. Liczniki zostały podzielone przez producenta na 6 kategorii:

- zaawanswane (ang. *advanced control*) – 16-bitowy;
- ogólnego zastosowania (ang. *general purpose*) – 16/32-bitowe;
- podstawowe (ang. *basic*) –16-bitowy;
- energooszczędne (ang. *low-power*) – 16-bitowy;
- licznik *SysTick* – 24-bitowy;
- liczniki dla watchdoga – 2 liczniki.

W grupie liczników podstawowych (*basic*) znajdziemy 2 moduły: TIM6 i TIM7. Sygnałem, który trafia na wejście tych liczników, jest zawsze sygnał zegarowy, którego częstotliwość może zostać podzielona za pomocą 16-bitowego *preskalera* (czyli częstotliwość może być dzielona przez wartości od 1 do 65 536). Aktualna wartość licznika jest przechowywana w rejestrze *CNT counter*, który również jest 16-bitowy. Wartość tego rejestru jest zwiększana podczas każdego cyklu (częstotliwość zegara podzielona przez *preskaler*), a następnie porównywana z zawartością rejestru *auto-reload*. Jeśli wartości te będą równe, to generowane jest przerwanie lub zdarzenie, a sam licznik rozpoczyna działanie od zera.

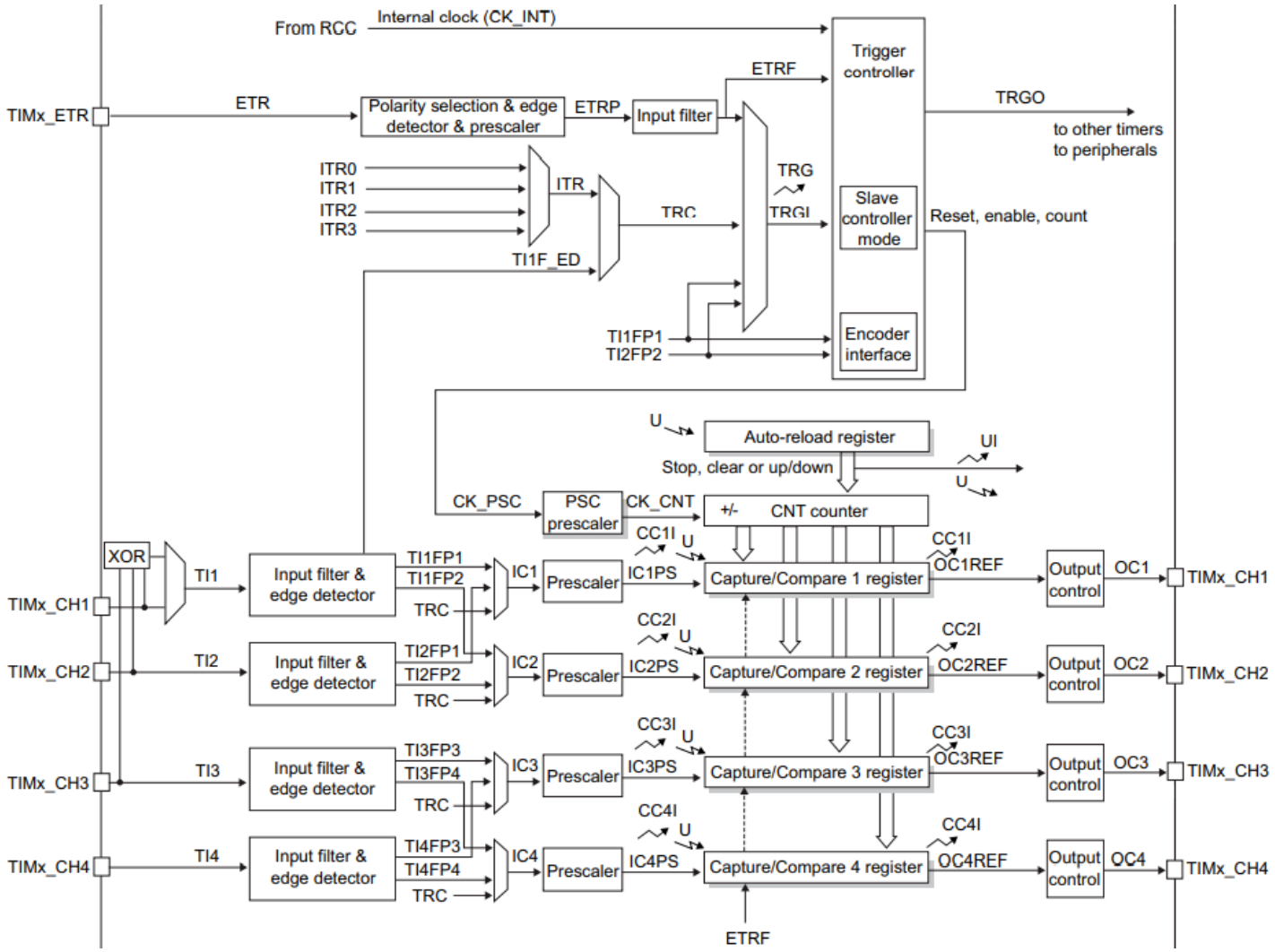


Liczniki ogólnego zastosowania (ang. *general purpose*) są bardziej rozbudowane. Oprócz samego licznika dodane zostały moduły kanałów przechwytywania/porównywania (ang. capture/compare). W przypadku konfiguracji ich jako wejście mogą one zapisywać stan licznika *CNT counter* w momencie wykrycia zdarzenia. Podczas pracy jako wyjście wartość w kanale jest porównywana z stanem licznika, a wynik porównania może sterować odpowiednim wyjściem. Ponadto licznik może zliczać impulsy nie tylko zegarowe, ale też pochodzące z zewnątrz (TIMx_ETR). W układzie STM32L496ZGT6 dostępne są liczniki TIM2, TIM3, TIM4 i TIM5 oraz o licznik z 2 kanałami TIM15 i 1 kanałem TIM16/TIM17.

1.4.2. Konwertery analogowo-cyfrowe[2]

Konwertery analogowo-cyfrowe (ang. *Analog-to-Digital Converter* - ADC) to układ służący do zamiany sygnału analogowego na sygnał cyfrowy. Dzięki temu możliwe jest jego przetwarzanie w systemach cyfrowych. Proces

[1] Opracowano na podstawie: https://forbot.pl/blog/kurs-stm32l4-liczniki-sprzetowe-pwm-enkoder-id46585
[2] Opracowano na podstawie: https://pl.wikipedia.org/wiki/Przetwornik_analogowo-cyfrowy

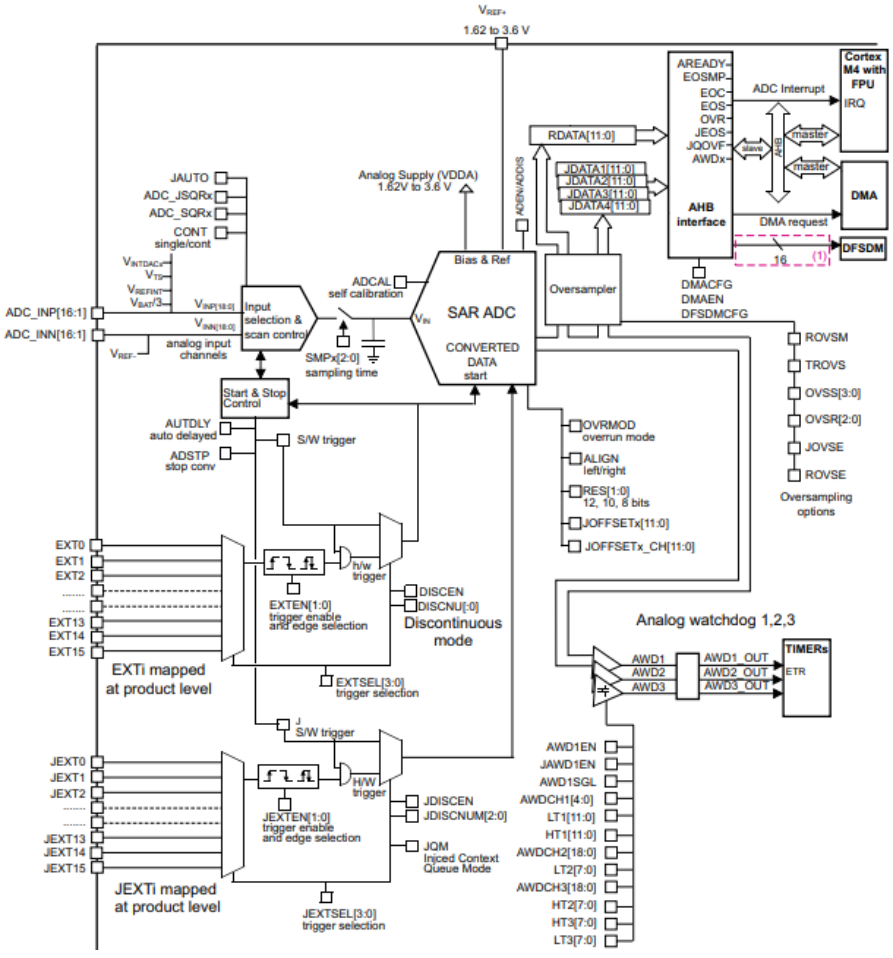


ten polega na uproszczeniu sygnału analogowego do postaci skwantowanej (dyskretnej), czyli zastąpieniu wartości zmieniających się płynnie do wartości zmieniających się skokowo w odpowiedniej skali (dokładności) odwzorowania. Przetwarzanie A/C tworzą 3 etapy: *próbkowanie*, *kwantyzacja* i *kodowanie*. Rozdzielczość przetwornika określa liczbę dyskretnych wartości jakie może on wytworzyć. Zwykle wyraża się ją w bitach. Przykładowo, przetwornik A/C, który potrafi przetworzyć próbkę sygnału na jedną z 256 wartości liczbowych posiada rozdzielczość równą 8 bitów. Rozdzielczość przetwornika może być również wyrażona w jednostkach napięcia. Rozdzielczość napięciowa przetwornika A/C jest równa jego całkowitej skali pomiaru (zakresowi) podzielonej przez liczbę poziomów kwantyzacji. Analogowy sygnał jest ciągły w czasie, więc konieczne jest przetworzenie go na ciąg liczb. To, jak często sygnał jest sprawdzany i zamieniany na liczbę zależną od jego poziomu, określane jest mianem *częstotliwości próbkowania* (f_s). Zwykle nie jest możliwe odtworzenie dokładnie takiego samego sygnału na podstawie wartości liczbowych, ponieważ dokładność jest ograniczona przez błąd kwantyzacji. Jednak wiarygodne odwzorowanie sygnału jest możliwe do osiągnięcia, gdy częstotliwość próbkowania jest równa co najmniej podwojonej najwyższej częstotliwości składowej sygnału (twierdzenie o próbkowaniu). Ze względu na metodę działania wyróżnia się trzy podstawowe metody pracy: bezpośrednia, pośrednia i kompensacyjna.

Przetwornik o przetwarzaniu bezpośrednim (ang. *flash*) działa na zasadzie bezpośredniego i zazwyczaj jednoczesnego porównania wartości napięcia wejściowego z szeregiem napięć odniesienia reprezentujących poszczególne poziomy kwantowania za pomocą szeregu komparatorów analogowych. Rezultat tego porównania wprowadzany jest na specjalny konwerter kodu, który wyprowadza wartość cyfrową sygnału

wejściowego w stosownej formie binarnej. Podstawową zaletą takich przetworników jest szybkość działania. Uzyskiwane szybkości przetwarzania są nawet od kilku razy do kilku rzędów wielkości większe od pozostałych typów przetworników A/C. Duża szybkość pracy okupiona jest relatywnie małą rozdzielczością oraz dokładnością. Zwiększenie rozdzielczości o kolejny bit wymaga podwojenia ilości elementów i zwiększenia precyzyjności napięć odniesienia uzyskiwanych zazwyczaj z dzielnika rezystorowego wysokostabilnego napięcia odniesienia wewnętrznego lub zewnętrznego. Dodatkowo zwiększanie ilości komparatorów, czyli poziomów kwantowania lub inaczej rozdzielczości bitowej przetwornika, powoduje zwiększenie jego pojemności wejściowej a co za tym idzie ograniczenia pasma wejściowego sygnału co niekorzystnie wpływa na jego parametry funkcjonalne. Przetworniki tego typu stosowane są wszędzie tam, gdzie wymagana jest bardzo duża częstotliwość próbkowania i jednocześnie nie jest wymagana bardzo duża dokładność przetwarzania (najczęściej nie większa niż 8- lub 9-bitowa).

Przetwornik z próbkowaniem analogowym (ang. *ramp-compare*) działa na zasadzie zliczania impulsów z generatora wzorcowego o dużej częstotliwości (względem czasu pomiaru) w czasie proporcjonalnym do napięcia wejściowego. Czas zliczania impulsów jest szerokością impulsu bramkującego generowanego przez układ sterujący na podstawie porównania napięcia wejściowego z liniowo narastającym napięciem odniesienia przez komparator analogowy. Parametry tego typu przetwornika bardzo mocno zależą od jakości (dokładności) generowania napięcia odniesienia (jest to przebieg piłokształtny) jego liniowości oraz powtarzalności szybkości narastania, a także od stabilności generatora wzorcowego. Szybkość przetwarzania, czyli częstotliwość próbkowania w tego typu przetwornikach równa jest częstotliwości przebiegu z generatora napięcia odniesienia. Uzyskiwane rozdzielczości zależą od szerokości bitowej licznika i częstotliwości generatora wzorcowego. Ze względu na to, iż stosunkowo łatwo jest zaprojektować i wytworzyć cyfrowe liczniki binarne o znacznych szerokościach słowa oraz generatory wzorcowe o dobrych parametrach, można w ten sposób uzyskać bardzo dobrą rozdzielczość przetwornika ograniczoną wyłącznie parametrem stosunku sygnału do szumu (SNR) części analogowej.



Przetwornik z sukcesywną aproksymacją (ang. *Successive Approximation* - SAR) działa na zasadzie porównywania wartości napięcia wejściowego z napięciem odniesienia wytworzonym za pomocą przetwornika cyfrowo-analogowego w iteracyjnym procesie obsługiwanym przez układ sterujący. Algorytm działania układu sterującego polega na ustawianiu (wartość „1") kolejnych bitów słowa danych dla przetwornika C/A poczynając od najważniejszego bitu słowa (MSB) i w przypadku, kiedy napięcie wejściowe

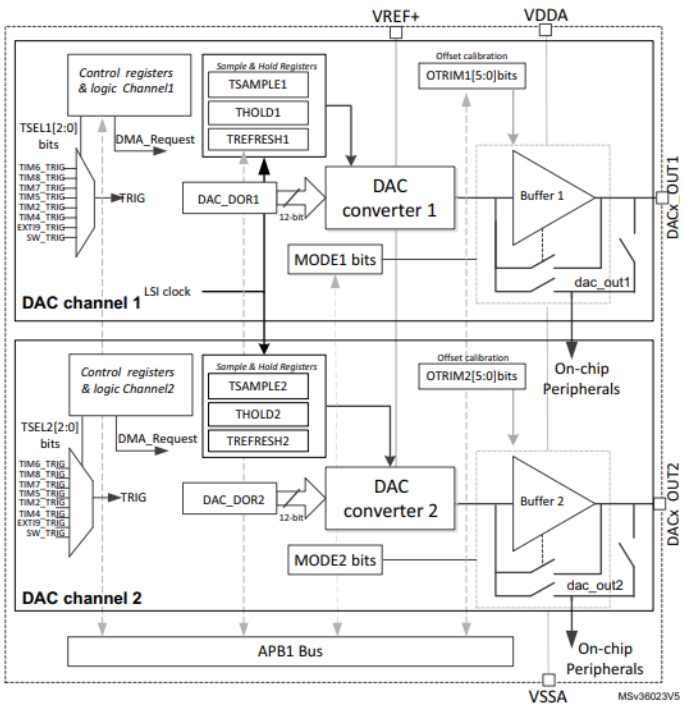
będzie mniejsze od napięcia odniesienia z przetwornika C/A to dany bit słowa danych jest kasowany (wartość „0”) w przeciwnym wypadku jest pozostawiany (wartość „1”) i realizowana jest kolejna iteracja algorytmu aż do osiągnięcia ostatniego bitu słowa danych (LSB). Tak ustawione słowo danych jest reprezentacją cyfrową napięcia wejściowego. Ze względu na iteracyjny charakter pracy przetwornika jego częstotliwość próbkowania jest znacząco mniejsza od uzyskiwanej w przetwornikach o przetwarzaniu bezpośrednim i w znacznym stopniu zależy od wielkości słowa danych – rozdzielczości przetwornika, szybkości pracy przetwornika C/A i w końcu komparatora i układu sterującego.

W mikrokontrolerze STM32L496ZGT6 dostępne są 3 konwertery 12-bitowe z sukcesywną aproksymacją: ADC1, ADC2 i ADC3. Każdy przetwornik może mierzyć sygnał przez 20 multipleksowanych wejść. Obsługiwane są różne tryby pracy: pojedyncza konwersja, ciągła konwersja, z automatycznym przełączaniem kanałów. Wynik konwersji może być zapisywany przez bezpośredni dostęp do pamięci DMA. Ponadto dodatkowe komparatory umożliwiają monitorowanie czy wynik pomiaru znajduje się w oczekiwanym zakresie (ang. *analog watchdog* - AWD). W celu poprawienia dokładności pomiaru można włączyć sprzętowe nad próbkowanie (ang *oversampling*) od 2 do 256 próbek. Rozdzielczość pomiaru może być ustawiona na 6, 8, 10 lub 12 bitów. Można porównywać napięcia pomiędzy wejściami. Rozpoczęcie konwersji może być programowe lub sprzętowe za pomocą licznika lub GPIO (łącznie 16 źródeł wyzwolenia pomiaru).

1.4.3. Konwerter cyfrowo-analogowy[3]

Przetwornik cyfrowo-analogowy (ang. *Digital to Analog Converter* - DAC) – układ elektroniczny przetwarzający sygnał cyfrowy w postaci liczby binarnej na sygnał analogowy w postaci prądu elektrycznego lub napięcia o wartości proporcjonalnej do tej liczby. Innymi słowy jest to układ przetwarzający dyskretny sygnał cyfrowy na równoważny mu sygnał analogowy. Podstawowe elementy przetwarzające sygnał cyfrowy na analogowy to:

- rejestr stanu, będący oddzielną częścią, który może być zintegrowany z zespołem przełączników, a w przetwornikach równoległych może w ogóle nie występować;
- zespół przełączników elektronicznych, sterowanych wejściowymi sygnałami cyfrowymi, każdemu bitowi odpowiada jeden przełącznik;
- sieć rezystorów;
- precyzyjne źródło napięcia odniesienia lub wejście do podłączenia takiego źródła.



Konstrukcyjne przetworniki C/A można podzielić na: uśredniające i mnożące. W układzie uśredniającym informacja na wejściu ma postać ciągu impulsów lub innego przebiegu o pewnej częstotliwości, można zastosować konwersje częstotliwości na napięcie. Przy bezpośrednim przetwarzaniu częstotliwości na napięcie w każdym okresie przebiegu zostaje wytworzony

[3] Opracowano na podstawie: https://pl.wikipedia.org/wiki/Przetwornik_cyfrowo-analogowy

standardowy impuls. Może to być impuls napięciowy bądź prądowy. Otrzymany ciąg impulsów zostaje uśredniony przez filtr dolnoprzepustowy lub integrator, co daje na wyjściu napięcie proporcjonalne do średniej częstotliwości sygnału wejściowego. W mnożących przetwornikach DAC wielkość wyjściowa jest iloczynem wejściowego napięcia i wejściowego kodu liczbowego.

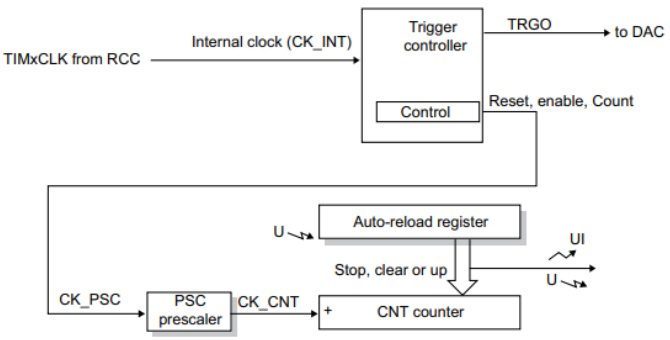
W mikrokontrolerze STM32L496ZGT6 dostępny jest jeden moduł dwukanałowego konwertera DAC o rozdzielczości 12 bitów i maksymalnej częstotliwości przetwarzania (f_s) 1 MHz. Długość słowa może być ustawiona na 12 lub 8 bitów. Transmisja danych może być realizowana z użyciem bezpośredniego dostępu do pamięci DMA. Konwersja może być wyzwolona programowo lub sprzętowo z użyciem licznika. Ponadto dostępny jest generator sygnału szumowego i trójkątnego o konfigurowanych parametrach.

1.4.4. Liczniki sprzętowe[4]

Odliczanie czasu, generowanie sygnału PWM i pomiar czasu trwania impulsu to tylko przykłady zadań do których używane są liczniki (ang. *timer*). W dużym skrócie można powiedzieć, że liczniki to moduły wewnątrz mikrokontrolerów, które zliczają „jakieś" impulsy i sygnalizują fakt doliczenia do ustalonej przez nas wartości. Mogą one zliczać takty sygnału zegarowego lub np. występowanie konkretnego zbocza na wejściu układu. W wyniku działania liczników mogą być generowane przerwania, ale mogą też dziać się inne rzeczy. Co ważne, te wszystkie operacje realizowane są sprzętowo, więc dzieją się niejako w tle i nie obciążają układu. Mikrokontrolery STM32 znane są m.in. właśnie z rozbudowanych liczników. Liczniki zostały podzielone przez producenta na 6 kategorii:

- zaawansowane (ang. *advanced control*) – 16-bitowy;
- ogólnego zastosowania (ang. *general purpose*) – 16/32-bitowe;
- podstawowe (ang. *basic*) –16-bitowy;
- energooszczędne (ang. *low-power*) – 16-bitowy;
- licznik SysTick – 24-bitowy;
- liczniki dla watchdoga – 2 liczniki.

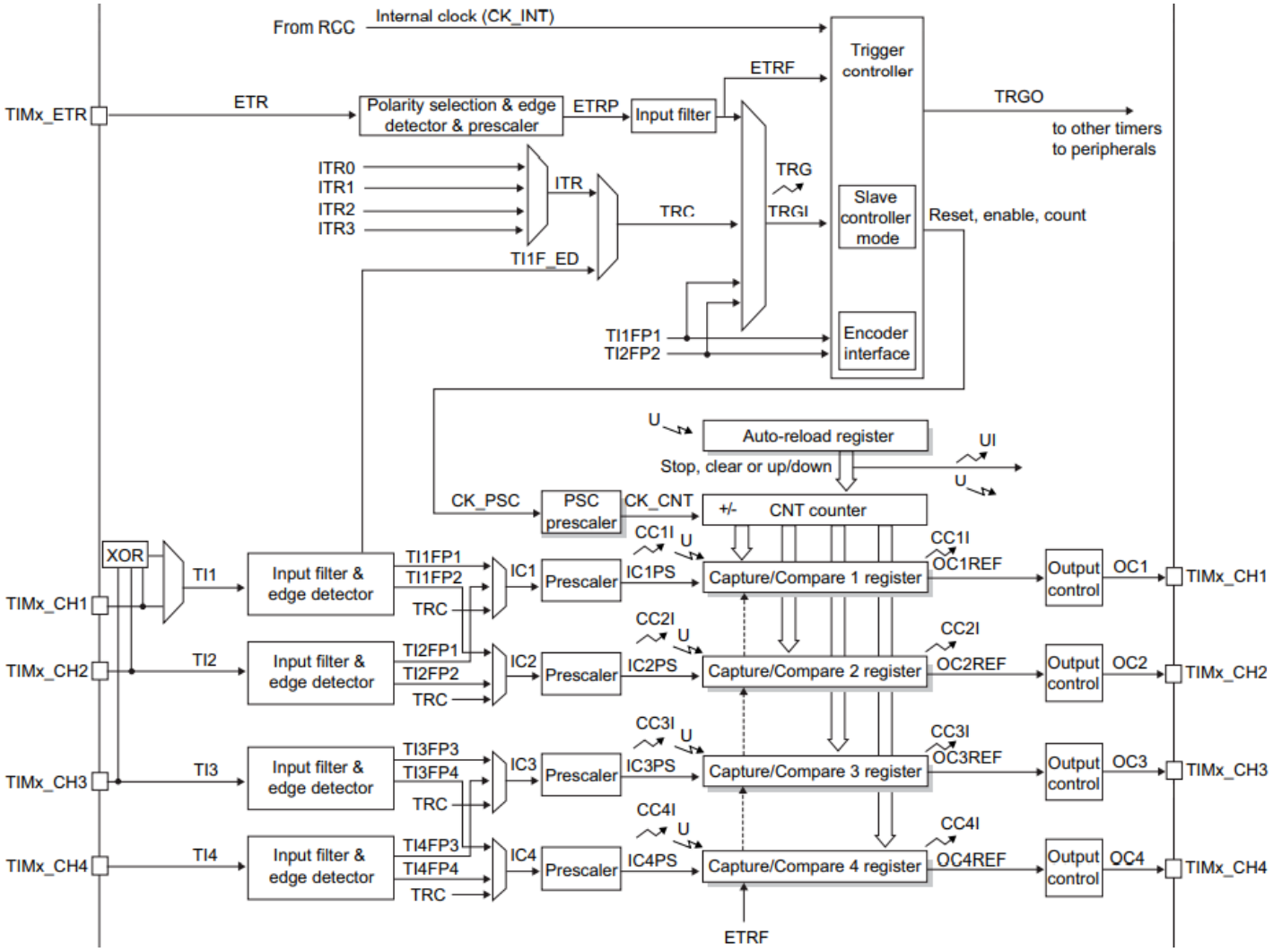
W grupie liczników podstawowych (*basic*) znajdziemy 2 moduły: TIM6 i TIM7. Sygnałem, który trafia na wejście tych liczników, jest zawsze sygnał zegarowy, którego częstotliwość może zostać podzielona za pomocą 16-bitowego *preskalera* (czyli częstotliwość może być dzielona przez wartości od 1 do 65 536). Aktualna wartość licznika jest przechowywana w rejestrze *CNT counter*, który również jest 16-bitowy. Wartość tego rejestru jest zwiększana podczas każdego cyklu (częstotliwość zegara podzielona przez *preskaler*), a następnie porównywana z zawartością rejestru *auto-reload*. Jeśli wartości te będą równe, to generowane jest przerwanie lub zdarzenie, a sam licznik rozpoczyna działanie od zera.



Liczniki ogólnego zastosowania (ang. *general purpose*) są bardziej rozbudowane. Oprócz samego licznika dodane zostały moduły kanałów przechwytywania/porównywania (ang. capture/compare). W przypadku

[4] Opracowano na podstawie: https://forbot.pl/blog/kurs-stm32l4-liczniki-sprzetowe-pwm-enkoder-id46585

konfiguracji ich jako wejście mogą one zapisywać stan licznika *CNT counter* w momencie wykrycia zdarzenia. Podczas pracy jako wyjście wartość w kanale jest porównywana z stanem licznika, a wynik porównania może sterować odpowiednim wyjściem. Ponadto licznik może zliczać impulsy nie tylko zegarowe, ale też pochodzące z zewnątrz (TIMx_ETR). W układzie STM32L496ZGT6 dostępne są liczniki TIM2, TIM3, TIM4 i TIM5 oraz o licznik z 2 kanałami TIM15 i 1 kanałem TIM16/TIM17.



1.5. Sposób realizacji ćwiczenia laboratoryjnego

Każde zajęcia składać będą się z następujących elementów:

- zadań obowiązkowych - do wykonania krok po kroku na podstawie instrukcji – do uzyskania od 0 do 12 pkt.;
- zadań uzupełniających – do samodzielnego zbudowania i napisania programu – do uzyskania od 0 do 14 pkt;
- pytań sprawdzających rozumienie działania programu – zadawane przez prowadzącego przyjmującego wykonanie zadania – odpowiedź wpływa na punktację z zadań obowiązkowych i uzupełniających, tzn. czy przyznać maksymalną możliwą liczbę punktów za zadanie czy tylko część przy ewidentnym braku zrozumienia problemu.

Po zrealizowaniu każdego z zadań należy poprosić prowadzącego o sprawdzenie i przyznanie punktów. Na koniec zajęć wystawiana jest ocena na podstawie sumy uzyskanych punktów: **2,0** <0; 10), **3,0** <10; 13), **3,5** <13; 16), **4,0** <16; 19), **4,5** <19; 23), **5,0** <23; 26>. W każdych zajęciach należy uczestniczyć. Przy każdym zadaniu

określona jest liczba punktów jakie można uzyskać. W przypadku zadań uzupełniających premiowana jest **jakość** i **czas realizacji** zaprezentowanego rozwiązania. Ocena końcowa z laboratorium to średnia arytmetyczna ocen z każdego spotkania.

W trakcie wykonywania ćwiczenia należy zwrócić szczególną uwagę na wyróżnione w treści instrukcji fragmenty tekstu wyróżnione kolorem:

- ***zielonym*** – etykiety wartości, nazwy pola, nazwy funkcji;
- ***niebieskim*** – wartości jakie należy użyć we skazanym miejscu;
- ***purpurowy*** – odwołania do kodu programu, nazw własnych.

2. Zadania podstawowe do realizacji (12 pkt.)

W tej części instrukcji zamieszczone są treści, z którymi obowiązkowo należy się zapoznać i praktycznie przećwiczyć. Ważne jest, aby zapamiętać wykonywane przedstawione czynności, aby móc na kolejnych zajęciach wykonywać je na kolejnych zajęciach bez potrzeby sięgania do niniejszej instrukcji.

Uwaga: Załączone wycinki z ekranu są poglądowe i pomagają jedynie w wskazaniu lokalizacji elementów interfejsu. Należy używać wartości podanych w tekście.

2.1. Zagadnienia wstępne

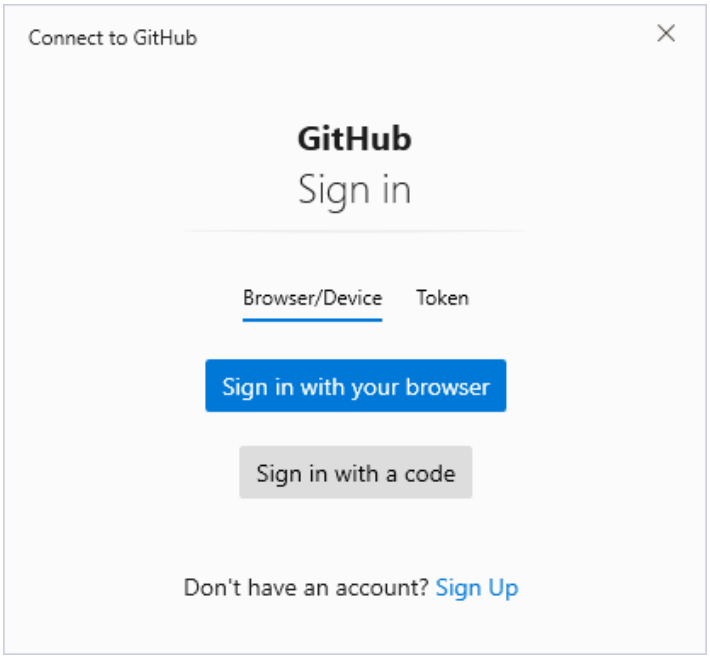
Przed przystąpieniem do pracy należy skonfigurować zainstalowane oprogramowanie i uzyskać dostęp do repozytorium, tzn. system kontroli wersji *Git*, instalacji biblioteki do obsługi mikrokontrolerów rodziny STM32L4 w środowisku *STM32CubeIDE* oraz sklonować repozytorium dla zajęć.

2.1.1. Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania

Na zajęciach należy dołączyć do wirtualnej grupy w ramach *Classrom GitHub* *za pomocą udostępnionego odnośnika prowadzącego do zadania*. Odnośnik prowadzi do kreatora w którym tworzone jest zadanie – indywidualne repozytorium studenta. Z listy należy wybrać swój adres e-mail i potwierdzić przyjęcie zadania (ang. *assignment*). Automatycznie zostanie utworzone prywatne repozytorium indywidualnie dla każdego studenta na podstawie przygotowanego repozytorium-szablonu. Na pierwszych zajęciach jest to wersja minimalna, a na kolejnych będzie zawierała już wstępnie skonfigurowane projekty. W sytuacji jeżeli student nie może odszukać się na liście (np. ktoś inny podłączył się pod daną osobę) proszę zgłosić to prowadzącemu zajęcia. W celu skorygowania nieprawidłowości. Zaakceptowanie zadania wiąże się również z dołączeniem do organizacji – wirtualnego konta organizacji w ramach którego tworzone są indywidualne repozytoria do zadań, z którego prowadzący zajęcia mają dostęp do wszystkich repozytoriów tworzonych w ramach zajęć.

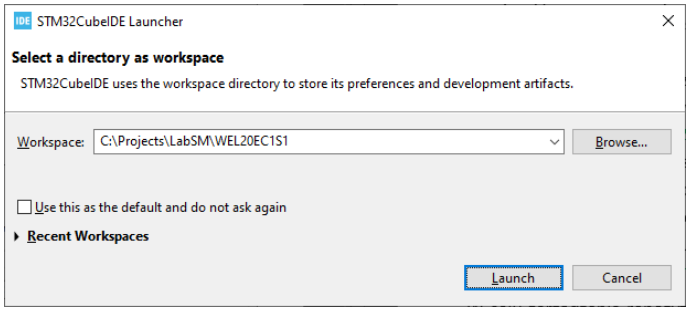
2.1.2. Pobranie repozytorium i konfiguracja lokalna

W celu pobrania repozytorium należy odszukać na pulpicie skrót o nazwie ***LabGitConfig***, który uruchomi skrypt wiersza poleceń, w którym należy podać: 1) *swoje imię i nazwisko*, 2) *adres e-mail*, 3) *symbol grupy*, 4) *numer ćwiczenia*, 5) *adres indywidualnego repozytorium* uzyskany w wcześniejszym punkcie. Po wykonaniu punktu 5) pojawi się okno logowania do serwisu *GitHub* podobne do zamieszczonego obok. W celu połączenia należy wybrać przycisk ***Sign in with your browser*** co spowoduje otworzenie nowej karty przeglądarki w którym należy udzielić dostępu do konta. Po uzyskaniu zgody nastąpi sklonowanie projektu z serwera do lokalizacji wynikającej z symbolu grupy. Na ten moment należy zminimalizować okno wiersza poleceń. Na zakończenie zajęć pozwoli ono na wypchnięcie zmian (ang. *push*) do repozytorium zdalnego. Przykładowy przebieg wykonania skryptu zamieszczony został poniżej.

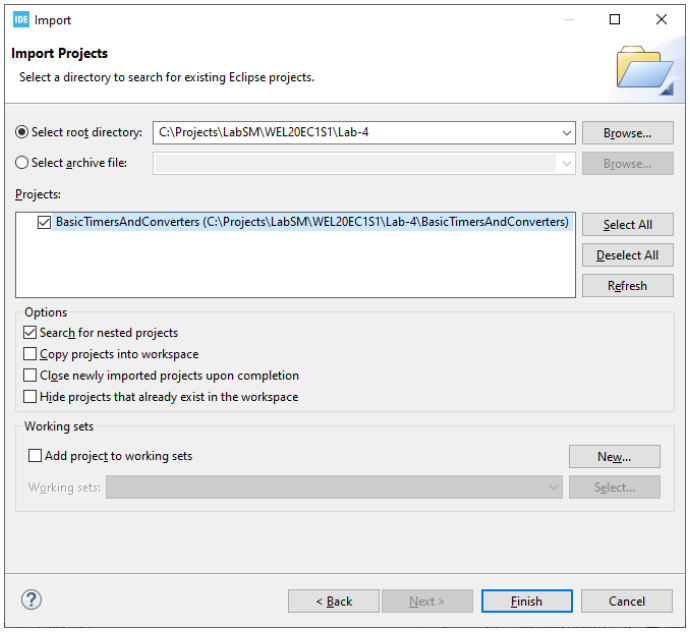


2.1.3. Konfiguracja środowiska STM32CubeIDE

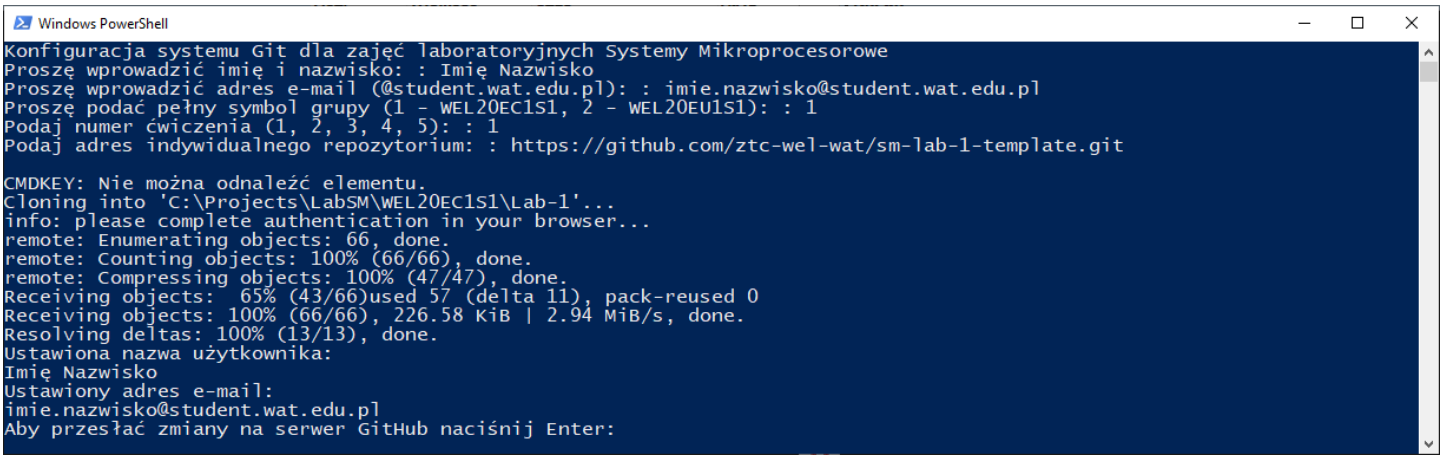
Korzystając ze skrótu na pulpicie lub z menu *Start* należ uruchomić środowisko projektowe *STM32CubeIDE*. Po jego uruchomieniu pojawi się pytanie o wskazanie katalogu, który będzie pełnił rolę przestrzeni roboczej. Należy wskazać na katalog *swojej grupy studenckiej* gdzie umieszczone zostało sklonowane repozytorium (w przykładzie: *C:\Projects\LabSM\WEL20EC1S1*). Wybór katalogu zatwierdzamy przyciskiem ***Launch***. Można używać wielu różnych przestrzeni roboczych w przypadku pracy z różnymi projektami. Po załadowaniu środowiska należy zamknąć zakładkę *Information Center*.



W kolejnym kroku należy zaimportować projekt do przestrzeni roboczej poprzez wybranie z menu ***File → Import***, a następnie w oknie, które się pojawi wybrać ***General → Existing Projects into Workspace***. Zatwierdzić przyciskiem ***Next***. Za pomocą przycisku ***Browse…*** wybrać należy katalog repozytorium (np. *C:\Projects\LabSM\WEL20EC1S1\Lab-4*). Zaktualizuje się lista dostępnych projektów i na niej należy wybrać ***BasicInterfacesHAL*** i zatwierdzić przyciskiem ***Finish***.



W celu zarządzania repozytorium dla zadania należy w środowisku *STM32CubeIDE* należy otworzyć widok (ang. *perspective*) zarządzania repozytorium. W tym celu z menu wybieramy: ***Window → Perspective → Open Perspective → Other…***, gdzie z listy należy wybrać ***Git***. Środowisko przełączy się do widoku *Git* i po lewej stronie pojawi się zakładka *Git Repositories*. Repozytorium dla zajęć zostanie automatycznie dodane do listy dostępnych repozytoriów. Przed wykonywaniem migawek należy się upewnić, że jest zaznaczone właściwe repozytorium, tj. ***Lab-4***.



W trakcie zajęć ograniczymy się do prostej liniowej struktury migawek wykonywanych po zakończeniu każdego z zadań instrukcji opatrzonych stosownym komentarzem. W dalszej części instrukcji będą podane treści komentarzy jakimi należy opatrzyć realizowane migawki. Powstanie zatem coś w rodzaju sprawozdania z zajęć, które będzie *przechowywane* w serwisie *GitHub*.

2.1.4. Opis projektu bazowego

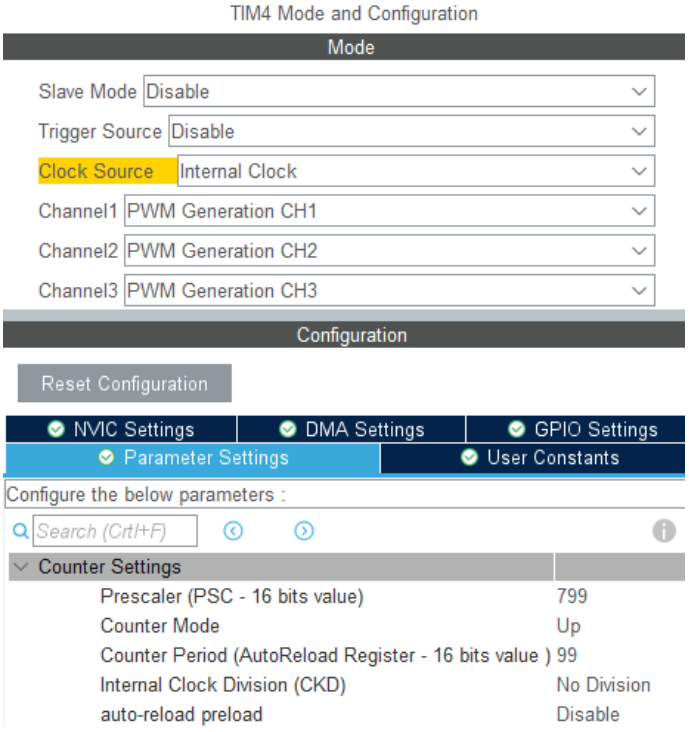
W sklonowanym repozytorium znajduje się projekt bazowy korzystający z biblioteki HAL o nazwie ***BasicTimersAndConverters***, który w poprzednim kroku został zaimportowany do środowiska *STM32CubeIDE*. Posiada on skonfigurowane wyprowadzenia mikrokontrolera do których dołączone są diody ***LED*** (*LED0* … *LED7*) oraz pięć styków joysticka joystick (*SW_RIGHT*, *SW_ LEFT*, *SW_ DOWN*, *SW_UP*, *SW_OK*). W pliku ***gpio.c*** zdefiniowane zostały podstawowe funkcje do obsługi diod LED oraz odczytu stanu joysticka. Ponadto skonfigurowana i dodana została obsługa wyświetlacza 7-segmentowego oraz tekstowego LCD w plikach ***display.c*** i ***lcd.c***. W pliku ***sensor.c*** dodana została obsługa czujników temperatury i przyspieszenia.

2.2. Konfiguracja i użycie liczników sprzętowych (6 pkt.)

Na płycie *KAmeLeon* do wyprowadzeń ***PD12***, ***PD13*** i ***PB8*** podłączona jest dioda ***LED RGB***. Wyprowadzenia te mogą być sterowane przez licznik ogólnego przeznaczenia ***TIM4***, który wytwarzać może przebiegi prostokątne o konfigurowanej częstotliwości, polaryzacji i wypełnieniu. Również złącza joysticka mogą posłużyć jako wejścia dla modułów licznika.

2.2.1. Sprzętowe generowania sygnału PWM (3 pkt.)

W celu skonfigurowania modułu TIM4 należy w projekcie otworzyć plik ***BasicTimersAndConverters.ioc*** w celu otworzenie okna konfiguratora. Należy w widoku ***Pinout & Configuration*** rozwinąć ***Timers*** i wybrać ***TIM4***. Następnie po prawej stronie w oknie ***TIM4 Mode and Configuration*** w sekcji ***Mode*** w polu wyboru źródła zliczanych impulsów (***Clock Source***) należy wybrać ***Internal Clock***. Następnie dla każdego z kanałów (***Channel1***, ***Channel1***, ***Channel3***) wybrać odpowiednio ***PWM Generation CH1***, ***PWM Generation CH2***, ***PWM Generation CH3***. Poniżej w sekcji ***Configuration*** zakładce ***Parameter Settings*** należy skonfigurować w grupie ***Counter Settings***: preskaler (***Prescaler***) ***799***, okres licznika (***Counter Period***) ***99***. Pozostałe opcje na tej zakładce pozostawiamy bez zmian. W związku z tym, że do licznika doprowadzony jest bazowy zegar o częstotliwości ***80 MHz***, to po podzieleniu w preskalerze częstotliwość zostanie zmniejszona do ***100 KHz***. Licznik zliczać będzie w górę od 0 do 99 czyli łącznie 100 impulsów w związku z czym będzie się resetował z częstotliwością ***1 KHz***. Poziom wypełnienia sygnału zależy od wartości ***Pulse*** konfigurowanej dla każdego kanału oddzielnie. Wprowadzone zmiany należy zapisać w celu wygenerowania kodu.



Następnie w funkcji ***main*** należy zmodyfikować kod pomiędzy znacznikami /* USER CODE BEGIN WHILE */ a /* USER CODE END WHILE */ jak to przedstawione poniżej. W pierwszej kolejności deklarujemy zmienną ***i***, która będzie użyta do przełączania kanałów w pętli ***for***. Następnie w tablicy ***PWM_Channel*** umieszczone są symbole identyfikująca kolejne kanały licznika. Zmienna ta jest użyta w funkcjach związanych z obsługą licznika. Pierwsza pętla ***for*** uruchamia poszczególne kanały licznika za pomocą funkcji ***HAL_TIM_PWM_Start***. W pętli

nieskończonej ***while*** odbywa się co sekundę losowanie nowej wartości dla kolejnych kanałów. Służy do tego funkcja ***rand*** dostępna w bibliotece ***stdlib.h***. Wynik jej działania jest ograniczony do przedziału od **0** do **99** za pomocą operacji dzielenia modulo. Ustawienie wypełnienia odbywa się za pomocą makra ***__HAL_TIM_SET_COMPARE***. Wartość ustawionego wypełnienia jest prezentowana na wyświetlaczu 7-segmentowym, a diody ***LED0…2*** informują dla którego kanału. Na koniec wywoływana jest funkcja ***HAL_Delay*** dla odczekania 1000 ms.

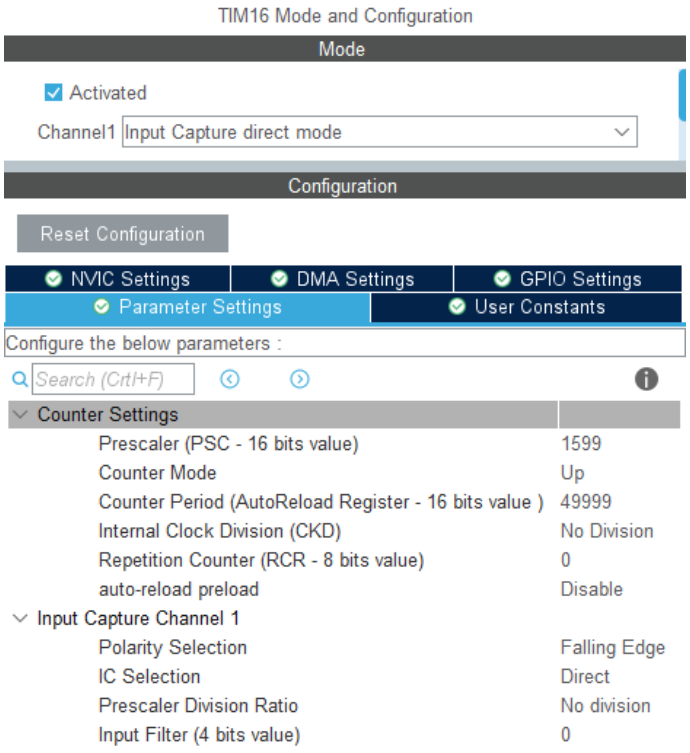
```
/* USER CODE BEGIN WHILE */
uint32_t i, duty;
uint16_t PWM_Channel[] = { TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3 };
for (i = 0; i < 3; i++)
    HAL_TIM_PWM_Start(&htim4, PWM_Channel[i]);

while (1) {
    for (i = 0; i < 3; i++) {
        duty = (uint16_t)(rand()%100);
        __HAL_TIM_SET_COMPARE(&htim4, PWM_Channel[i], duty);
        SEG_DisplayDec(duty);
        LEDs_SetValue(1 << i);
        HAL_Delay(1000);
    }
}
/* USER CODE END WHILE */
```

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia do ocenienia. Po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.2.1 – sprzętowe generowania sygnału PWM***”.

2.2.2. Rejestrowanie momentu wystąpienia zdarzeń z użyciem trybu wejścia licznika (3 pkt.)

W celu zmiany konfiguracji projektu należy w projekcie otworzyć plik ***BasicTimersAndConverters.ioc***. Należy odszukać wyprowadzenie ***PE0*** mikrokontrolera do którego jest podłączony styk ***SW_RIGHT*** joysticka i zmienić tryb jego pracy z ***GPIO_Input*** na ***TIM16_CH1*** *nazwę pinu należy zachować*. Należy w widoku ***Pinout & Configuration*** rozwinąć ***Timers*** i wybrać ***TIM16***. Następnie po prawej stronie w oknie ***TIM16 Mode and Configuration*** w sekcji ***Mode*** zaznaczyć pole ***Activated*** w celu podłączenia do zegara taktującego licznik ***TIM16***. Następnie dla kanału ***Channel1*** wybrać ***Input Capture direct mode***. Poniżej w sekcji ***Configuration*** zakładce ***Parameter Settings*** należy skonfigurować w grupie ***Counter Settings***: preskaler (***Prescaler***) ***1599***, okres licznika (***Counter Period***) ***49999***. W grupie ***Input Capture Channel 1*** należy zmienić wykrywaną polaryzację (***Polarity Selection***) z ***Rising Edge*** na ***Falling Edge***. Pozostałe opcje na tej zakładce pozostawiamy bez zmian. Następnie należy przejść do zakładki ***NVIC Settings*** i zaznaczyć pole ***Enabled*** dla ***TIM1 update interrupt and TIM16 global interrupt***. Wprowadzone zmiany należy zapisać w celu wygenerowania kodu. W związku z tym, że do licznika doprowadzony jest bazowy zegar



o częstotliwości **80 MHz**, to po podzieleniu w preskalerze częstotliwość zostanie zmniejszona do **50 KHz**. Licznik zliczać będzie w górę od **0** do **49999** czyli łącznie 50000 impulsów w związku z czym będzie się resetował z częstotliwością **1 Hz**. Oznacza to, że moment wystąpienia zdarzenia opadającego zbocza na wyprowadzeniu ***PE0*** będzie rejestrowany z dokładnością 20 µs.

Na potrzebę przechowywania znacznika czasu należy zdefiniować typ struktury ***TimeStamp_t***, która będzie składać się z trzech pozycji: ***second***, ***milisecond*** oraz ***microsecond***. Należy ją zdefiniować w sekcji kodu pomiędzy znacznikami /* USER CODE BEGIN PTD */ a /* USER CODE END PTD */ jak to przedstawione poniżej.

```
/* USER CODE BEGIN PTD */
typedef struct{
    uint16_t second;
    uint16_t milisecond;
    uint16_t microsecond;
}TimeStamp_t;
```

W sekcji kodu znajdującej się pomiędzy znacznikami /* USER CODE BEGIN PV */ a /* USER CODE END PV */ należy zdefiniować zmienną: znacznika czasu ***timeStamp***, przechowującą liczbę sekund jaka upłynęła od uruchomienia licznika ***second*** oraz flagę informującą o utworzeniu nowego znacznika czasu ***update***.

```
/* USER CODE BEGIN PV */
TimeStamp_t timeStamp;
uint16_t second = 0;
FlagStatus update = RESET;
/* USER CODE END PV */
```

Następnie w funkcji *main* należy zmodyfikować kod pomiędzy znacznikami /* USER CODE BEGIN WHILE */ a /* USER CODE END WHILE */ jak to przedstawione poniżej. W pierwszej kolejności tworzymy tablicę ***lcd*** do przechowywania łańcucha znaków reprezentującego znacznik czasowy. Później inicjowany jest wyświetlacz LCD i włączane jego podświetlenie. Za pomocą funkcji ***HAL_TIM_Base_Start_IT*** uruchamiany jest licznik, który został skonfigurowany wcześniej tak, aby przepełniał się co 1 sekundę. Przepełnienie ma generować przerwanie, które obsługuje się w funkcji ***HAL_TIM_PeriodElapsedCallback***. Kolejna funkcja (***HAL_TIM_IC_Start_IT***) uruchamia przechwytywanie stanu licznika w momencie wykrycia zdarzenia oraz generuje przerwanie, które obsługuje się w funkcji ***HAL_TIM_IC_CaptureCallback***. W pętli nieskończonej ***while*** w sytuacji kiedy zmienna ***update*** jest ustawiona następuje: konwersja znacznika czasu do sformatowanego łańcucha znaków z użyciem funkcji ***sprintf***, aktualizacja zawartości wyświetlacza oraz skasowanie zmiennej ***update***.

```
/* USER CODE BEGIN WHILE */
char lcd[16];
LCD_Init();
LCD_BacklightOn();
HAL_TIM_Base_Start_IT(&htim16);
HAL_TIM_IC_Start_IT(&htim16, TIM_CHANNEL_1);
while (1) {
    if (update) {
        sprintf(lcd, "%d.%03d.%03d", (int) timeStamp.second,
                (int) timeStamp.milisecond, (int) timeStamp.microsecond);
        LCD_WriteTextXY((uint8_t*) lcd, strlen(lcd), 1, 1);
        update = RESET;
    }
}
/* USER CODE END WHILE */
```

Następnie należy dodać funkcję obsługi przerwań od licznika pomiędzy znacznikami /* USER CODE BEGIN 4 */ a /* USER CODE END 4 */ jak to przedstawione poniżej. W funkcji ***HAL_TIM_PeriodElapsedCallback*** sprawdzane jest najpierw czy przerwanie zostało spowodowane przez właściwy licznik, a następnie jeżeli warunek jest spełniony zwiększany jest licznik sekund. Druga funkcja ***HAL_TIM_CaptureCallback*** jest bardziej rozbudowana. Oprócz sprawdzenia który moduł jest źródłem przerwania, sprawdzamy również który kanał. Jest to ważne w przypadku równoległego korzystania z więcej niż jednego kanału w danym liczniku. Za pomocą

funkcji ***HAL_TIM_ReadCapturedValue*** odczytujemy zapamiętany stan licznika i mnożymy go przez 20 µs aby uzyskać wynik w mikrosekundach. W dalszej części dokonujemy wpisu do pozycji struktury ***timeStamp***. Na zakończenie ustawiana jest zmienna ***update***.

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM16) {
        // Po każdym przepełnieniu licznika zwiększenie liczby sekund
        second++;
    }
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM16) {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
            // Odczytanie zatrzaśniętego stanu licznika
            uint32_t microsecond = HAL_TIM_ReadCapturedValue(htim,
                    TIM_CHANNEL_1) * 20;
            // Zapisanie znacznika na podstawie stanu licznika
            timeStamp.second = second;
            timeStamp.milisecond = (uint16_t) (microsecond / 1000);
            timeStamp.microsecond = (uint16_t) (microsecond % 1000);
            update = SET;
        }
    }
}
```

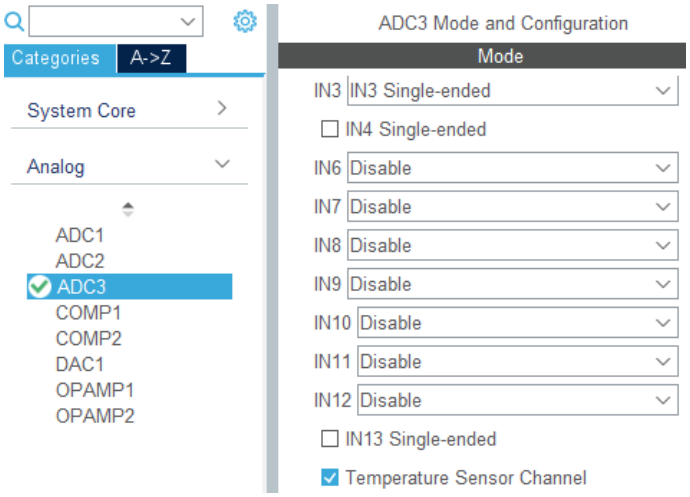
Przygotowany program należy skompilować, uruchomić i przetestować odchylając joystick w prawo obserwując zawartość wyświetlacza LCD. Działanie programu przedstawić prowadzącemu zajęcia do ocenienia. Po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.2.2 – rejestrowanie momentu wystąpienia zdarzeń z użyciem trybu wejścia licznika***”.

2.3. Obsługa przetwornika ADC (6 pkt.)

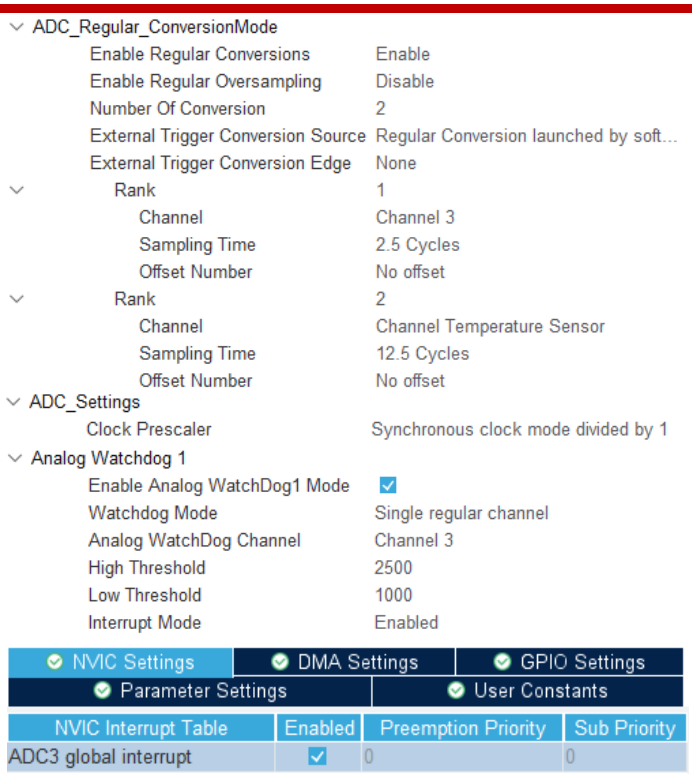
Na płycie *KAmeLeon* zamontowany został w prawym górnym rogu potencjometr obrotowy, którego odczep podłączony jest do wyprowadzenia ***PC2*** mikrokontrolera, a do wyprowadzenia ***PC3*** doprowadzony jest sygnał z wzmacniacza mikrofonu analogowego. Ponadto sam mikrokontroler umożliwia monitorowanie wewnętrznych napięć oraz dysponuje wbudowanym czujnikiem temperatury.

2.3.1. Monitorowanie położenia potencjometru i temperatury układu (3 pkt.)

W zadaniu tym należy skonfigurować moduł przetwornika ***ADC3*** tak aby dokonywał pomiaru sygnału od wspomnianych wyżej elementów oraz monitorował czy napięcie na potencjometrze mieści się w wskazanym przedziale. W celu skonfigurowania modułu *ADC3* należy w projekcie otworzyć plik ***BasicTimersAndConverters.ioc*** w celu otworzenie okna konfiguratora. Należy odszukać wyprowadzenie ***PC2*** i skonfigurować je jako ***ADC3_IN3***. Następnie należy w widoku ***Pinout & Configuration*** rozwinąć ***Analog*** i wybrać ***ADC3***. Następnie po prawej stronie w oknie ***ADC3 Mode and Configuration*** w sekcji ***Mode*** należy zmienić dla ***IN3*** z ***Disable*** na ***IN3 Single-ended*** oraz zaznaczyć ***Temperature Sensor Channel***.



Poniżej w sekcji ***Configuration***, zakładce ***Parameter Settings*** należy skonfigurować w grupie ***ADC_Settings*** dzielnik zegara taktującego moduł (***Clock Prescaler***) ***Synchronus clock mode divided by 1***. W grupie ***ADC_Regular_ConversionMode*** zmienić ***Number Of Conversion*** z ***1*** na ***2***, a następnie dla ***Rank 2*** zmienić ***Channel*** *na* ***Channel Temperature Sensor*** oraz zmienić ***Sampling Time*** *z* ***2.5 Cycles*** na ***12.5 Cycles***. W grupie ***Analog Watchdog 1*** zaznaczyć ***Enable Analog WatchDog Mode***, ustawić ***High Threshold*** na ***2500*** i ***Low Threshold*** na ***1000*** oraz zmienić ***Interrupt mode*** z ***Disabled*** na ***Enabled***. Następnie należy otworzyć zakładkę ***NVIC Settings*** i zaznaczyć ***Enabled*** dla ***ADC3 global interrupt***. Wprowadzone zmiany należy zapisać w celu wygenerowania kodu.



Wygenerowany kod w pliku *Core->Src->main.c* wymaga uzupełnienia o funkcjonalność pozwalającą na prezentacje wyników pomiaru. W tym celu należy dodać w pierwszej kolejności zmienną globalną ***analogWatchdogStatus*** typu ***uint32_t*** w sekcji /* USER CODE BEGIN PV */ a /* USER CODE END PV */.

```
/* USER CODE BEGIN PV */
uint32_t analogWatchdogStatus = RESET;
/* USER CODE END PV */
```

Następnie w funkcji ***main*** należy zmodyfikować kod pomiędzy znacznikami /* USER CODE BEGIN WHILE */ a /* USER CODE END WHILE */ jak to przedstawione poniżej.

```
/* USER CODE BEGIN WHILE */
uint32_t i = 0;
uint16_t adcConversionValue[] = { 0, 0 };

if (HAL_ADCEx_Calibration_Start(&hadc3, ADC_SINGLE_ENDED) != HAL_OK)
    Error_Handler();

while (1) {
    if (HAL_ADC_Start(&hadc3) != HAL_OK)
        Error_Handler();
    // Oczekiwanie na zakończenie pomiaru napięcia na potencjometrze
    while (__HAL_ADC_GET_FLAG(&hadc3, ADC_FLAG_EOC) != SET)
        ;
    adcConversionValue[0] = HAL_ADC_GetValue(&hadc3);
    // Oczekiwanie na zakończenie pomiaru napięcia na czujniku temperatury
    while (__HAL_ADC_GET_FLAG(&hadc3, ADC_FLAG_EOC) != SET)
        ;
    adcConversionValue[1] = HAL_ADC_GetValue(&hadc3);
    // Naprzemienne wyświetlanie zmierzonej wartości
    SEG_DisplayDec(adcConversionValue[i & 1]);
    LEDs_SetValue((1 << (i++ & 1)) | (analogWatchdogStatus << 7));
    HAL_Delay(1000);
    analogWatchdogStatus = RESET;
}
/* USER CODE END WHILE */
```

W pierwszej kolejności definiowane są dwie zmienne: ***i*** oraz ***adcConversionValue***. Pierwsza służy do wybierania, która wartość z drugiej zmiennej ma być wyświetlona na wyświetlaczu 7-segmentowym. Druga

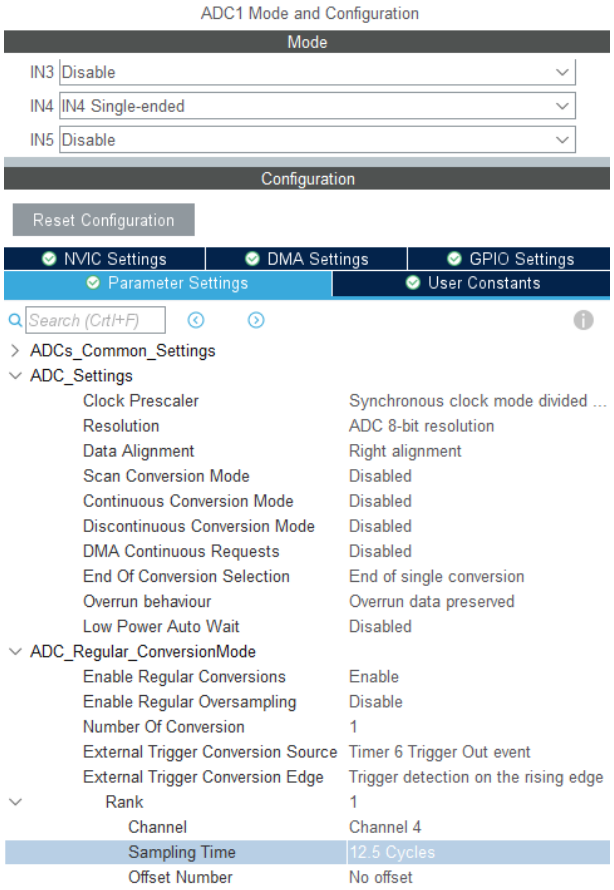
zmienna w postaci tablicy przechowuje wynik pomiaru dla potencjometru [0] i czujnika temperatury [1]. Funkcja ***HAL_ADCEx_Calibration_Start*** przeprowadza jednorazową kalibracje przetwornika, a w przypadku wystąpienia błędu przekieruje program do funkcji ***Error_Handler***. Cyklicznie do 1000 ms w pętli *while* odbywa się: 1) uruchomienie konwersji (***HAL_ADC_Start***); 2) oczekiwanie na zakończenie konwersji; 3) zapisanie wyniku konwersji do tablicy; 4) oczekiwanie na zakończenie konwersji; 5) zapisanie wyniku konwersji do tablicy; 6) wyświetlanie na przemian wyników konwersji na wyświetlaczu 7-segmentowym; 7) prezentacja na diodach LED informacji o tym która wartość jest prezentowana (***LED0*** lub ***LED1***) oraz czy wartość zmierzona dla potencjometru mieści się ustawionym w zakresie (***LED7*** włączona kiedy poza zakresem); 8) odczekanie 1000 ms; 9) skasowanie zmiennej ***analogWatchdogStatus***. Informacja o tym, że wynik pomiaru napięcia na potencjometrze nie mieści się w określonym zakresie ***<1000; 2500>*** jest komunikowana za pomocą przerwania i należy je obsłużyć. W tym celu w sekcji pomiędzy znacznikami /* USER CODE BEGIN 4 */ a /* USER CODE END 4 */ należy dodać implementację funkcji obsługi zdarzenia ***HAL_ADC_LevelOutOfWindowCallback***. Funkcja ta jest automatycznie wywoływana przez bibliotekę HAL i ustawia zmienną ***analogWatchdogStatus***.

```
/* USER CODE BEGIN 4 */
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc) {
    analogWatchdogStatus = SET;
}
```

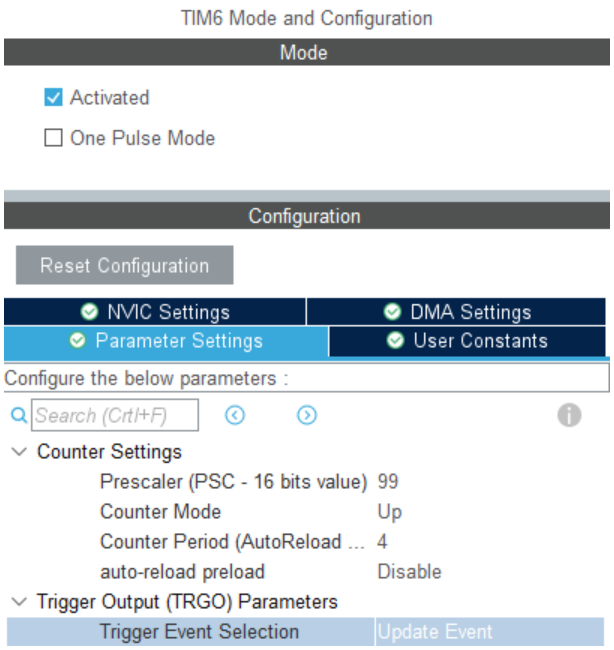
Przygotowany program należy skompilować, uruchomić i przetestować. Zmieniając położenie potencjometru zaobserwować sygnalizację wyjścia poza przedział przez zapalenie się diody ***LED7***. Działanie programu przedstawić prowadzącemu do ocenienie. Po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.3.1 – monitorowanie położenia potencjometru i temperatury układu***”.

2.3.2. Podłączenie mikrofonu analogowego do komputera (3 pkt.)

W zadaniu tym należy skonfigurować moduł przetwornika ***ADC1*** tak aby dokonywał pomiaru sygnału na wejściu ***PC3***. W celu skonfigurowania modułu *ADC1* należy w projekcie otworzyć plik ***BasicTimersAndConverters.ioc*** w celu otworzenie okna konfiguratora. Należy odszukać wyprowadzenie ***PC3*** i skonfigurować je jako ***ADC1_IN4***. Następnie należy w widoku ***Pinout & Configuration*** rozwinąć ***Analog*** i wybrać ***ADC1***. Następnie po prawej stronie w oknie ***ADC1 Mode and Configuration*** w sekcji ***Mode*** należy zmienić dla ***IN4*** z ***Disable*** na ***IN4 Single-ended***. Poniżej w sekcji ***Configuration***, zakładce ***Parameter Settings*** należy skonfigurować w grupie ***ADC_Settings*** dzielnik zegara taktującego moduł (***Clock Prescaler***) ***Synchronus clock mode divided by 1***, zmienić rozdzielczość (***Resolution***) na ***ADC 8-bit resolution***. W grupie ***ADC_Regular_ConversionMode*** zmienić sposób wyzwalania pomiaru (***External Trigger Conversion Source***) na ***Timer 6 Trigger Out event***. W podgrupie ***Rank 1*** zmienić ***Channel*** na ***Channel 4*** oraz zmienić ***Sampling Time*** z ***2.5 Cycles*** na ***12.5 Cycles***. Następnie należy otworzyć



zakładkę ***NVIC Settings*** i zaznaczyć ***Enabled*** dla ***ADC1 global interrupt***. Pozostałe opcje na tej zakładce pozostawiamy bez zmian. Z racji, że przetwarzanie ma być uruchamiane przez licznik ***TIM6*** należy również go skonfigurować. W tym celu w widoku ***Pinout & Configuration*** rozwinąć ***Timers*** i wybrać ***TIM6***. Następnie po prawej stronie w oknie ***TIM6 Mode and Configuration*** w sekcji ***Mode*** zaznaczyć pole ***Activated*** w celu podłączenia do zegara taktującego licznik ***TIM6***. Poniżej w sekcji ***Configuration*** zakładce ***Parameter Settings*** należy skonfigurować w grupie ***Counter Settings***: preskaler (***Prescaler***) ***999***, okres licznika (***Counter Period***) ***9***. Pozostałe opcje na tej zakładce pozostawiamy bez zmian. Dalej poniżej w sekcji ***Trigger Event Selection*** dla pozycji źródła wyzwalania zdarzenia (***Trigger Event Selection***) ustawić ***Update Event***. W związku z tym, że do licznika doprowadzony jest bazowy zegar o częstotliwości **80 MHz**, to po podzieleniu w preskalerze częstotliwość zostanie zmniejszona do **800 KHz**. Licznik zliczać będzie w górę od **0** do **9** czyli łącznie 10 impulsów w związku z czym będzie się resetował z częstotliwością **8 kHz**. Pozostałe opcje na tej zakładce pozostawiamy bez zmian. Wprowadzone zmiany należy zapisać w celu wygenerowania kodu.



Następnie w funkcji ***main*** należy zmodyfikować kod pomiędzy znacznikami /* USER CODE BEGIN WHILE */ a /* USER CODE END WHILE */ jak to przedstawione poniżej. Zmiany sprowadzają się do wywołania funkcji ***HAL_TIM_Base_Start***, która uruchamia licznik wyzwalający przetwornik, a funkcja ***HAL_ADC_Start_IT*** włącza przetwornik w trybie zgłaszania przerwania na zakończenie konwersji.

```
/* USER CODE BEGIN WHILE */
  HAL_TIM_Base_Start(&htim6);
  HAL_ADC_Start_IT(&hadc1);
  while (1) {

  }
  /* USER CODE END WHILE */
```

W celu obsłużenia przerwania od zakończenia konwersji należy w sekcji pomiędzy znacznikami /* USER CODE BEGIN 4 */ a /* USER CODE END 4 */ należy dodać implementację funkcji obsługi zdarzenia ***HAL_ADC_ConvCpltCallback***. W niej odczytywany jest wynik konwersji, a następnie od razu wysyłany za pośrednictwem interfejsu szeregowego do komputera.

```
246 /* USER CODE BEGIN 4 */
247 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
248     if(hadc->Instance == ADC1){
249         int8_t sample = (int8_t) HAL_ADC_GetValue(&hadc1) - INT8_MAX;
250         HAL_UART_Transmit(&hlpuart1, (uint8_t*) &sample, 1, 10);
251     }
252 }
```

Przygotowany program należy skompilować, uruchomić i przetestować z użyciem programu *SerialPlot* obserwując przebieg rejestrowanego przez mikrofon sygnału. Działanie programu przedstawić prowadzącemu do ocenienia. Po zatwierdzeniu wykonać migawkę z komentarzem „***Zadanie 2.3.2 – rejestracja sygnału z mikrofonu analogowego***”.

3. Zadania rozszerzające do realizacji (14 pkt.)

W rozdziale tym przedstawione zostały zadania dodatkowe, za realizację których można podnieść ocenę końcową za wykonane ćwiczenie. Ich realizację należy wykonać w dotychczasowym projekcie.

3.1.1. Sterowanie wypełnieniem sygnału PWM przez czujnik przyspieszenia LSM303C (3 pkt.)

W zadaniu tym należy rozbudować funkcjonalność programu uzyskanej w zadaniu 2.2.1 w taki sposób, że wypełnienie sygnału PWM wytwarzanego przez licznik TIM4 zależy od wartości bezwzględnych odczytanych z czujnika przyspieszenia LSM303C. Oś X steruje kanałem 1, oś Y steruje kanałem 2, a oś Z steruje kanałem 1. Należy odczytane przyspieszenie przeskalować tak, że maksymalne przyspieszenie (tzn. 2g) oznacza wypełnienie 100%. Funkcję do inicjalizacji czujnika i odczytu danych o przyspieszeniu zostały zamieszczone w pliku *sensor.c*.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „***Zadanie 3.1.1 – sterowanie wypełnieniem sygnału PWM przez czujnik przyspieszenia LSM303C***”.

3.1.2. Rozbudowanie funkcjonalności znacznika czasu (4 pkt.)

W zadaniu tym należy rozbudować funkcjonalność programu uzyskanej w zadaniu 2.2.2 w taki sposób, że wyznaczana będzie różnica czasu pomiędzy kolejnymi zdarzeniami oraz prezentacja wyniku będzie uwzględniać minuty. W tym celu należy przechowywać dodatkową kopię znacznika czasu w celu wyznaczenia różnicy, którą należy wyświetlić w drugim wierszu wyświetlacza tekstowego LCD.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „***Zadanie 3.1.2 – rozbudowanie funkcjonalności znacznika czasu***”.

3.2. Rozszerzona obsługa przetwornika ADC (7 pkt.)

3.2.1. Konwersja wartości dyskretnej na rzeczywistą i prezentacja na wyświetlaczu LCD (3 pkt.)

W zadaniu tym należy rozbudować funkcjonalność programu uzyskanej w zadaniu 2.3.1 o prezentacje na wyświetlaczu tekstowym LCD w pierwszym wierszu napięcia mierzonego na potencjometrze, a w drugim wierszu temperatury odczytywanej z wbudowanego czujnika w mikrokontrolerze. Jako napięcie odniesienia dla przetwornika należy przyjąć 3,3 V. Sposób wyznaczania temperatury można znaleźć w dokumencie RM0351 w rozdziale 18.4.32. W celu zamiany liczby zmiennoprzecinkowej do postaci łańcucha znaków można użyć funkcji biblioteki języka C *sprintf* oraz funkcji *LCD_WriteTextXY* (dokumentacja umieszczona w pliku nagłówkowym *lcd.h*).

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „***Zadanie 3.1.1 – konwersja wartości dyskretnej na rzeczywistą i prezentacja na wyświetlaczu LCD***”.

3.2.2. Wyznaczenie parametrów rejestrowanego sygnału (4 pkt.)

W zadaniu tym należy rozbudować funkcjonalność programu uzyskanej w zadaniu 2.3.2 o możliwość wyznaczenia maksymalnej amplitudy sygnału wejściowego oraz średniej amplitudy sygnału za okres 1 sekundy oraz prezentacja uzyskanych wyników na wyświetlaczu tekstowym LCD.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „***Zadanie 3.2.2 – wyznaczenie parametrów rejestrowanego sygnału***”.