Sterowniki robotów

Laboratorium – Regulator PID

Implementacja i strojenie regulatora PID na bazie symulatora silnika

Wojciech Domski

Dokument `sr_lab04b.pdf`, Wersja 1.0.1

Spis treści

1 Wprowadzenie

Celem zadania jest zapoznanie się z sterowaniem obiektami z wykorzystaniem regulator PID w zamkniętej pętli zwrotnej. Jednym z najbardziej typowych zastosowań regulatora PID jest wykorzystanie go do sterowania prędkością obrotową silników. W ramach ćwiczenia silnik elektryczny został zastąpiony układem zastępczym – RC. Układ ten przypomina swoim działaniem silnik elektryczny.

2 Opis ćwiczenia

Ćwiczenie składa się z kilku etapów, w jego ramach należy

1. skonfigurować odpowiednie peryferia (ADC, DAC, TIM),
2. wyliczyć stałą czasową dla układu zastępczego,
3. zbadać odpowiedź skokową,
4. uzupełnić implementację regulatora PID,
5. dobrać nastawy regulatora.

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*. Przykłady 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

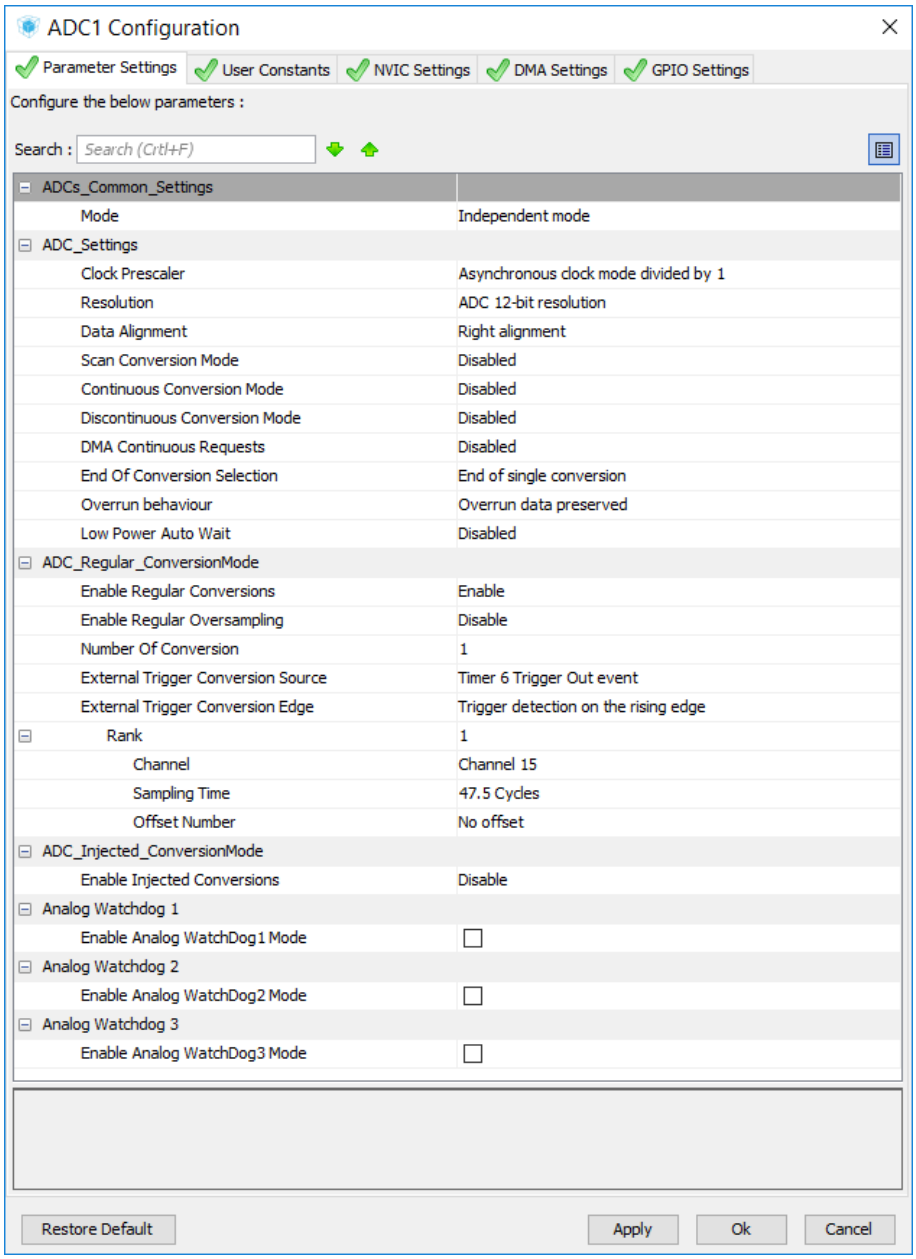
Uwaga! Pamiętaj, aby wyłączyć optymalizację kodu (-O0), a także wykorzystać kompilację równoległą (*parallel*) np. -j8 [3]. Ponadto, ustaw automatyczny zapis przed wykonaniem kompilacji, aby uniknąć problemów związanych z rozbieżnością kodu, a plikiem wynikowym wgranym na mikrokontroler. Pamiętaj również, 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*.

W ramach ćwiczenia należy zrealizować kilka etapów, które w efekcie końcowym doprowadzą do implementacji układu sterowania symulatorem silnika. Wszystkie niezbędne peryferia wykorzystywane podczas ćwiczenia znajdują się na płytce Nucleo [4].

Podczas ćwiczenia należy uzupełnić dostarczone fragmenty programu, tak, aby regulator wymuszał na obiekcie sterowania śledzenie zadanej trajektorii. W tym celu zadana trajektoria (funkcja sinusoidalna) została stabularyzowana. Częstotliwość funkcji okresowej może być regulowana za pomocą programowego dzielnika zegara *dac_ nperiod_ max*. Zmniejszenie tej wartości spowoduje, że częstotliwość generowanego sygnału ulegnie zwiększeniu. Parametr ten nie powinien być modyfikowany, chyba, że zajdzie taka konieczność np. inne wartości elementów układu symulatora obrotów silnika.

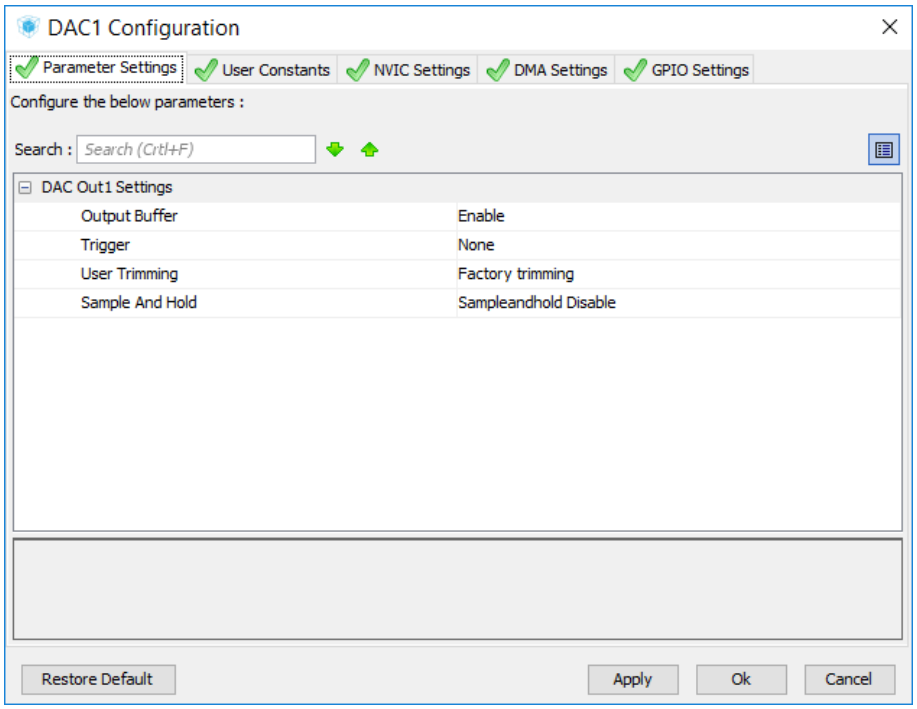
3.1 Konfiguracja peryferiów

W celu realizacji ćwiczenia należy skonfigurować trzy peryferia. Pierwszym z nich jest przetwornik ADC. Zostanie on wykorzystany do pomiaru stanu obiektu regulacji. Można wykorzystać dowolny z dostępnych kanałów ADC. Przykładowa konfiguracja została przedstawiona na rys. 1, jako kanał pomiarowy wybrano kanał 15 (pin PB0). Podczas ustawiania parametrów przetwornika należy zwrócić szczególną uwagę na konfigurację trybu działania, wybór odpowiedniego kanału oraz wyzwalanie pomiaru za pomocą zdarzenia pochodzącego od licznika. Pozwoli to na automatyzację procesu pomiarowego. Dodatkowo należy włączyć przerwanie dla tego peryferium.



Rysunek 1: Konfiguracja przetwornika ADC1 w programie CubeMX

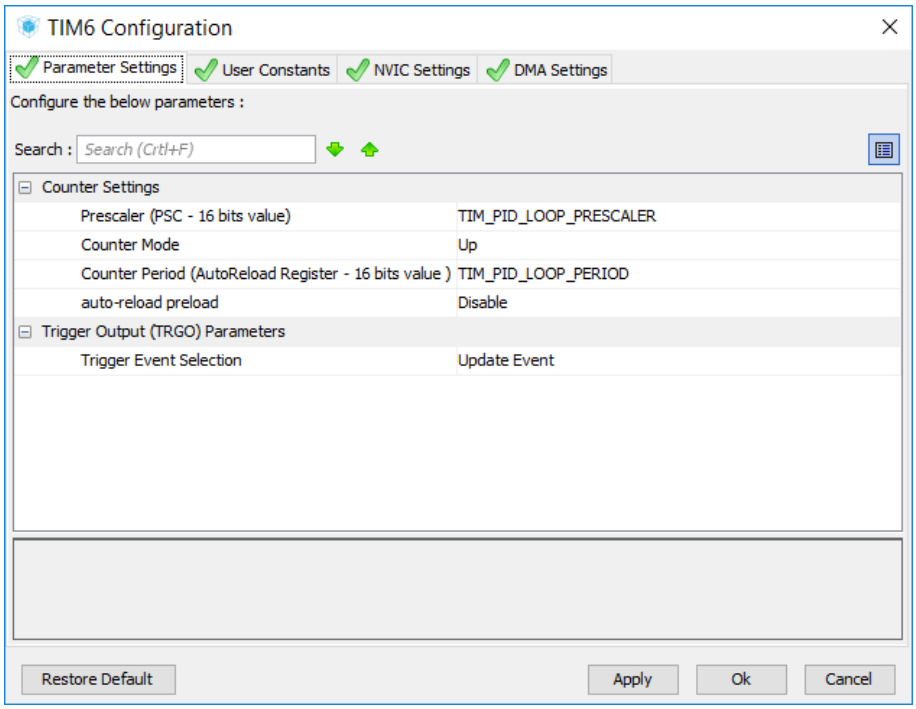
Kolejnym krokiem jest konfiguracja przetwornika cyfrowo-analogowego DAC. Również i tym razem przykładowa konfiguracja została przedstawiona na rys. 2.



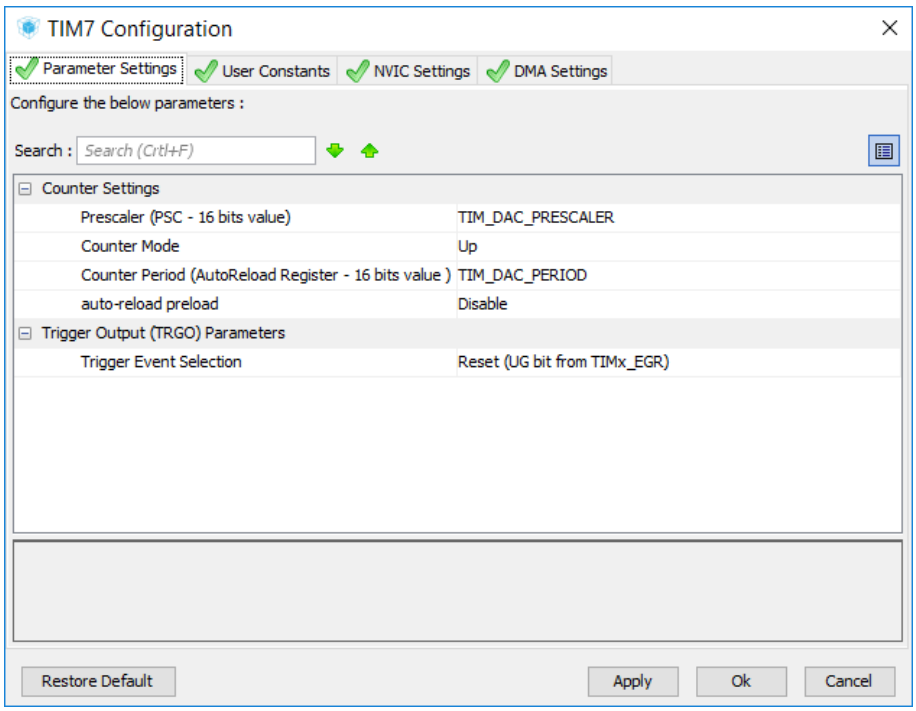
Rysunek 2: Konfiguracja przetwornika DAC1 w programie CubeMX

W przypadku układów liczących należy skonfigurować 2 układy liczące: TIM6 oraz TIM7. Ich konfiguracja została zaprezentowana na rys. 3 oraz rys. 4 odpowiednio. Licznik TIM6 pełni dwie funkcje. Pierwszą z nich jest generowanie podstawy czasu dla pętli sterowania – regulatora PID. Druga funkcja to automatyczne wyzwalanie pomiaru z wykorzystaniem przetwornika analogowo–cyfrowego. Z kolei układ TIM7 odpowiedzialny jest za generowanie odpowiedniej wartości zadanej dla układu regulacji.

Pętla regulacji PID powinna być wyzwalana cyklicznie co 1 ms. W tym celu należy odpowiednio skonfigurować licznik TIM6 poprzez ustalenie odpowiednich wartości *TIM_PID_LOOP_PRESCALER* oraz *TIM_PID_LOOP_PERIOD*. W przypadku licznika TIM7, który jest odpowiedzialny za generowania wartości zadanej poprzez pośrednie wybieranie wartości z tablicy LUT również należy go odpowiednio skonfigurować. W tym przypadku przerwania od licznika powinny być generowane z częstotliwością 1kHz, aby to osiągnąć należy odpowiednio zdefiniować stałe *TIM_DAC_PRESCALER* oraz *TIM_DAC_PERIOD* w programie `CubeMx`.



Rysunek 3: Konfiguracja licznika TIM6 w programie CubeMX



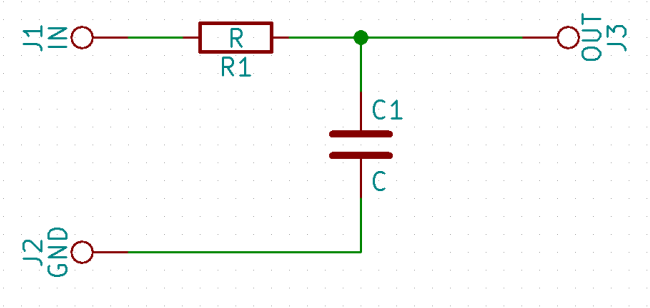
Rysunek 4: Konfiguracja licznika TIM7 w programie CubeMX

3.2 Wyliczenie stałej czasowej dla układu RC

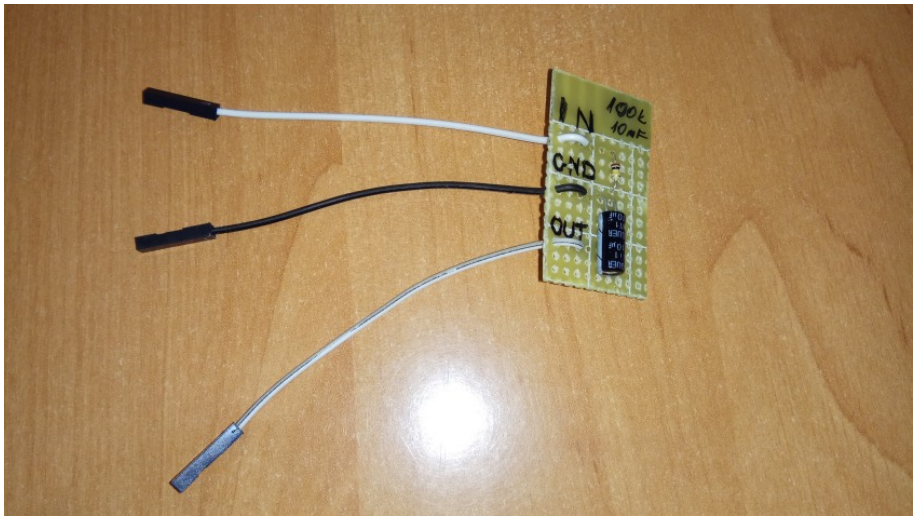
Na rys. 5 przedstawiono schemat układu RC. Jest to układ, który de facto może być wykorzystany, jako filtr dolnoprzepustowy. Należy wyliczyć stałą czasową dla tego układu. Każdy egzemplarz został opisany i zawiera informację na temat wykorzystanych komponentów, a w szczególności ich wartości. Zdjęcie przykładowego egzemplarza zostało zaprezentowane na rys. 6. Podczas obliczeń należy korzystać

ze wzoru na rozładowanie kondensatora C1 przez rezystor R1:

$$\tau = RC. \tag{1}$$



Rysunek 5: Schemat układu RC



Rysunek 6: Symulator obrotów silnika – układ RC

3.3 Podłączenie układu

Układ RC należy podłączyć zgodnie z poniższą tabelą:

Mikrokontroler	Układ RC
IN	PA4
OUT	PB0
GND	GND

Tablica 1: Połączenia pomiędzy wyjściami mikrokontrolera oraz układ RC

Podłączenie należy realizować wyłącznie przy wyłączonym zasilaniu. Po połączeniu płytki Nucleo z układem RC należy zgłosić ten fakt prowadzącemu. Bez pozytywnej decyzji prowadzącego o poprawnym połączeniu modułów nie można realizować dalszej części zadania.

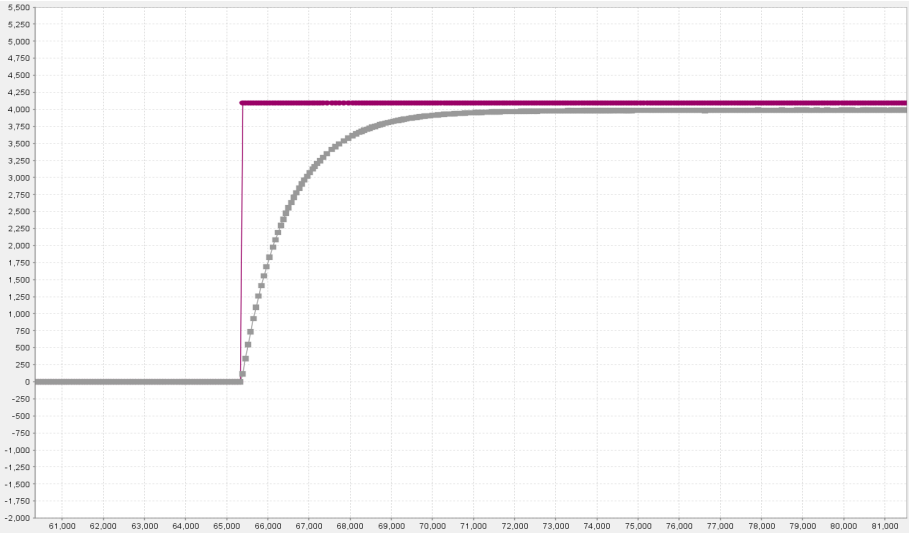
3.4 Badanie odpowiedzi skokowej

Podczas badania odpowiedzi skokowej należy zapewnić odpowiednie warunki – rozładowanie kondensatora C1 (rys. 5). Można to wykonać na różne sposoby, jednym z nich jest zadawanie odpowiednich sygnałów na wyjścia mikrokotrolera, do których podłączony jest układ. Poniższy fragment kodu powinien zapewnić odpowiednie warunki pracy oraz umożliwić badania odpowiedzi skokowej.


```
1  HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 0);
2  //rozładowanie kondensatora C1
3  HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);
4  HAL_Delay(100000);
5  //zadanie skoku na wejście układu IN
6  HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 4095);
7  HAL_Delay(100000);
```

W jaki inny sposób można bezpiecznie rozładować kondensator C1?

Odpowiedź skokową należy zarejestrować w programie `STMStudio`. Przykładowa wykres z odpowiedzią skokową symulatora silnika został przedstawiony na rys. 7.



Rysunek 7: Odpowiedź skokowa dla układu RC (R=100kOhm, C=10uF)

3.5 Implementacja regulatora PID

Regulator PID [1] składa się z trzech członów: proporcjonalnego (P), całkującego (I) oraz różniczkującego (D). Istnieje wiele pochodnych tego regulatora, jednakże, jego najczęściej spotykaną wersją jest postać opisana następującym równaniem:

$$u(t) = K_p e(t) + K_i \int_0^T e(t)dt + K_d \dot{e}(t), \tag{2}$$

gdzie błąd $e(t)$ definiowany jest jako różnica wartości zadanej i wartości zmierzonej, $e(t) = y_{sp}(t) - y(t)$. Dyskretna implementacja regulatora PID może przyjąć następującą postać:

$$u_k = K_p e_k + \tau K_i \sum_{i=0}^{k} e_i + \frac{1}{\tau} K_d \dot{e}_k, \tag{3}$$

gdzie τ to interwał z jakim wykonywana jest pętla regulacji, e_i to wartość uchybu regulacji w chwili czasowej i, natomiast $\dot{e}_k = e_k - e_{k-1}$. Przykładową implementację na procesory 8-bitowe można znaleźć w [2], gdzie zaprezentowano algorytm regulatora PID z wykorzystaniem stałego przecinka. Ponadto, wydajną implementację regulatora PID, jak i innych algorytmów można znaleźć w CMSIS DSP [5].

W załączniku A do dokumentu została zaprezentowana przykładowa implementacja, którą należy uzupełnić we wskazanych miejscach. Przedstawiony kod źródłowy jest implementacją regulatora PID z wykorzystaniem stałego przecinka, gdzie jego pozycja może być regulowana. Ponadto zastosowano tam ograniczenia maksymalnej oraz minimalnej wartości wszystkich poszczególnych członów regulatora, jak i ich sumy co pozwala na uniknięcie sytuacji, w której układy peryferyjne mikrokontrolera nie będą wstanie zrealizować sygnału sterującego. Do miejsc wymagających uzupełnienia należą wyliczenie:

- uchybu regulacji,
- sygnału sterującego pochodzącego od członu proporcjonalnego P,
- sygnału sterującego pochodzącego od członu całkującego I,

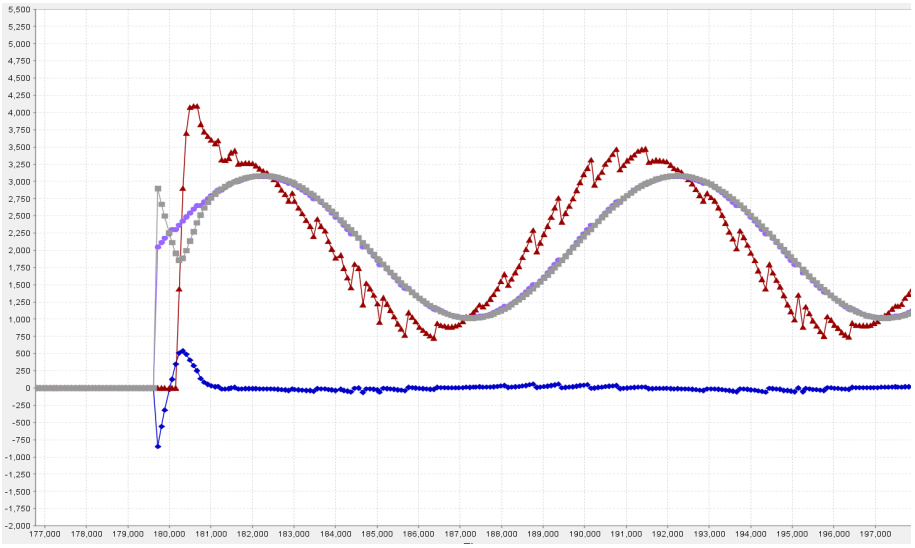
- sygnału sterującego pochodzącego od członu różniczkującego D.

Zmienne, które będą przydatne podczas uzupełniania kodu to:

- e – uchyb regulacji,
- *pid->e_last* – wartość uchyb regulacji w poprzedniej iteracji sterownika,
- dv – wartość zadana,
- mv – wartość zmierzona,
- p – sygnał sterujący pochodzący od członu proporcjonalnego,
- i – sygnał sterujący pochodzący od członu całkującego,
- d – sygnał sterujący pochodzący od członu różniczkującego,
- *pid->p* – wzmocnienie członu proporcjonalnego,
- *pid->i* – wzmocnienie członu całkującego,
- *pid->d* – wzmocnienie członu różniczkującego,
- *pid->dt_ms* – interwał pracy regulatora PID wyrażony w milisekundach.

Ponadto w załączniku B również zamieszczono fragmenty kodu, które stanowią implementację prostego sterownika dla symulatora regulatora obrotów silnika (układ RC). W kodzie, który został zaprezentowany należy dobrać parametry regulatora, aby uzyskać możliwe mały uchyb regulacji.

Przykładowy przebieg regulacji przedstawiono na rys. 8. Na wspomnianym rysunku szarą linią zaznaczono wartość zmierzoną na wyjściu układu RC (OUT, mv). Fioletową linią pokazano wartość zadaną (dv), natomiast czerwona linia reprezentuje sygnał sterujący dla układu RC (IN, *dac_control*). Wreszcie linią o kolorze niebieskim zaznaczono uchyb regulacji (e).



Rysunek 8: Sygnały w symulatorze regulatora obrotów silnika

3.6 Dobranie nastaw regulatora

Jak wspomniano wcześniej należy dobrać odpowiednie wartości parametrów regulatora. W tym celu należy modyfikować linię programu, która zawiera poniższą instrukcję:

```
1 pid_init(&pid, PPP, III, DDD, 10, 1);
```

Dobrymi wartościami początkowymi są *PPP = 1.0f*, *III = 0.0f*, *DDD = 0.0f*.

3.7 * Zadanie dodatkowe

Wykorzystaj układ licznika w trybie generatora sygnału PWM zamiast przetwornika cyfrowo-analogowego.

3.8 Uporządkowanie stanowiska

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

4 Podsumowanie

Ćwiczenie przedstawia kilka aspektów związanych z strojeniem regulatorów PID począwszy od konfiguracji peryferiów, aż po badanie charakterystyk obiektu oraz dobór parametrów regulatora. Przedstawiona została również problematyka związana z doborem adekwatnych peryferiów mikrokontrolera, które pozwalają na sterowanie układem, jak i badaniem jego stanu.

Literatura

[1] K. J. Äström, T. Hägglund. *PID Controllers: Theory, Design and Tuning.* wydanie 2.

[2] Atmel. *AVR221: Discrete PID controller.*

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

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

[5] ST. *Digital signal processing for STM32 microcontrollers using CMSIS.*, Marzec, 2016.

Załącznik A

```
/*
* pid.c
*
*  Created on: 09.03.2018
*      Author: Wojciech Domski
*/

#include "pid.h"

void pid_init(cpid_t * pid, float p, float i, float d, uint8_t f, int32_t dt_ms) {
    uint32_t k;

    pid->power = 1;
    for (k = 0; k < f; ++k) {
        pid->power = pid->power * 2;
    }
    pid->f = f;

    pid->p = (int32_t) (p * pid->power);
    pid->i = (int32_t) (i * pid->power);
    pid->d = (int32_t) (d * pid->power);

    pid->p_val = 0;
    pid->i_val = 0;
    pid->d_val = 0;

    pid->p_max = INT32_MAX;
    pid->p_min = INT32_MIN;

    pid->i_max = INT32_MAX;
    pid->i_min = INT32_MIN;

    pid->d_max = INT32_MAX;
    pid->d_min = INT32_MIN;

    pid->e_last = 0;
    pid->sum = 0;

    pid->total_max = INT32_MAX;
    pid->total_min = INT32_MIN;

    pid->dt_ms = dt_ms;
}
```

```
int32_t pid_calc(cpid_t * pid, int32_t mv, int32_t dv) {
    int32_t p, i, d, e, total;

    pid->mv = mv;
    pid->dv = dv;

    //UZUPELNIJ WYLICZANIE BLEDU
    e = ...;
    //UZUPELNIJ WYLICZANIE SYGNALU PRZEZ CZLON PROPORCJONALNY
    p = ...;
    if (p > pid->p_max)
        p = pid->p_max;
    else if (p < pid->p_min)
        p = pid->p_min;

    pid->p_val = p >> pid->f;

    i = pid->sum;
    //UZUPELNIJ WYLICZANIE SYGNALU PRZEZ CZLON CALKUJACY
    //PAMIETAJ O SKALOWANIU CZASU WYKONYWANIA PETLI DO SEKUND
    i += ...;
    if (i > pid->i_max)
        i = pid->i_max;
    else if (i < pid->i_min)
        i = pid->i_min;
    pid->sum = i;

    pid->i_val = i >> pid->f;

    //UZUPELNIJ WYLICZANIE SYGNALU PRZEZ CZLON ROZNICZKUJACEGO
    //PAMIETAJ O SKALOWANIU CZASU WYKONYWANIA PETLI DO SEKUND
    d = ...;
    if (d > pid->d_max)
    d = pid->d_max;
    else if (d < pid->d_min)
    d = pid->d_min;

    pid->d_val = d >> pid->f;

    total = p + i + d;

    if (total > pid->total_max)
        total = pid->total_max;
    else if (total < pid->total_min)
        total = pid->total_min;
    pid->control = total >> pid->f;
    pid->e_last = e;

    return pid->control;
}

int32_t pid_scale(cpid_t * pid, float v) {
    return v * pid->power;
}
```



```
/*
* pid.h
*
*  Created on: 09.03.2018
*      Author: Wojciech Domski
*/

#ifndef PID_H_
#define PID_H_

#include <stdint.h>

typedef struct {
    int32_t p;
    int32_t i;
    int32_t d;

    int32_t p_val;
    int32_t i_val;
    int32_t d_val;

    int32_t p_max;
    int32_t i_max;
    int32_t d_max;

    int32_t p_min;
    int32_t i_min;
    int32_t d_min;

    uint8_t f;
    uint32_t power;

    int32_t dv;
    int32_t mv;

    int32_t e_last;
    int32_t sum;

    int32_t total_max;
    int32_t total_min;

    int32_t control;

    int32_t dt_ms;
} cpid_t;

void pid_init(cpid_t * pid, float p, float i, float d, uint8_t f, int32_t dt_ms);

int32_t pid_calc(cpid_t * pid, int32_t mv, int32_t dv);

int32_t pid_scale(cpid_t * pid, float v);

#endif /* PID_H_ */
```


Załącznik B

Nagłówki, które należy dodać do programu

```
#include <stdio.h>
#include "pid.h"
```

Zmienne, które należy dodać do programu

```
volatile int adc_flag;
volatile int adc_value;
volatile int dac_value;
volatile int dac_control;
volatile uint16_t dac_index;
volatile uint8_t dac_nperiod;
#define dac_nperiod_max    100 //programowy prescaler
int step_desired;
cpid_t pid;

uint16_t sin_wave[] = { 2048, 2112, 2176, 2239, 2302, 2364, 2424, 2483, 2541,
        2596, 2649, 2700, 2748, 2794, 2836, 2876, 2912, 2945, 2974, 2999, 3021,
        3039, 3053, 3063, 3069, 3071, 3069, 3063, 3053, 3039, 3021, 2999, 2974,
        2945, 2912, 2876, 2836, 2794, 2748, 2700, 2649, 2596, 2541, 2483, 2424,
        2364, 2302, 2239, 2176, 2112, 2048, 1983, 1919, 1856, 1793, 1731, 1671,
        1612, 1554, 1499, 1446, 1395, 1347, 1301, 1259, 1219, 1183, 1150, 1121,
        1096, 1074, 1056, 1042, 1032, 1026, 1024, 1026, 1032, 1042, 1056, 1074,
        1096, 1121, 1150, 1183, 1219, 1259, 1301, 1347, 1395, 1446, 1499, 1554,
        1612, 1671, 1731, 1793, 1856, 1919, 1983, };
```

Dodaj następujące definicje funkcji zwrotnych do programu:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    if (hadc == &hadc1) {
        adc_flag = 1;
        adc_value = HAL_ADC_GetValue(&hadc1);
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim == &htim7) {
        dac_value = sin_wave[dac_index];

        ++dac_nperiod;
        if (dac_nperiod >= dac_nperiod_max) {
            dac_nperiod = 0;
            ++dac_index;
            if (dac_index >= 100)
                dac_index = 0;
        }
    }
}
```

Inicjalizacja oraz główna pętla programu

```
HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 0);
HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);

//PID LOOP
HAL_TIM_Base_Start(&htim6);
//DAC DESIRED VALUE
HAL_TIM_Base_Start_IT(&htim7);

HAL_ADC_Start_IT(&hadc1);

//DOBIERZ ODPOWIEDNIE WSPOLCZYNNIKI REGULATORA PID
pid_init(&pid, 1.0f, 0.0f, 0.0f, 10, 1);

pid.p_max = pid_scale(&pid, 4095);
pid.p_min = pid_scale(&pid, -4095);
pid.i_max = pid_scale(&pid, 4095);
pid.i_min = pid_scale(&pid, -4095);
pid.d_max = pid_scale(&pid, 4095);
pid.d_min = pid_scale(&pid, -4095);
pid.total_max = pid_scale(&pid, 4095);
pid.total_min = pid_scale(&pid, 0);

while (1) {
    if (adc_flag == 1) {
        adc_flag = 0;

        dac_control = pid_calc(&pid, adc_value, dac_value);

        HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R,
            dac_control);
    }
}
```

