SPRAWOZDANIE Z PROJEKTU		ROK AKADEMICKI 2021/22
Przedmiot: **SYSTEMY MIKROPROCESOROWE**		
Temat ćwiczenia: Regulacja temperatury rezystora z wykorzystaniem regulatora PID	**Nr zestawu:**	Termin zajęć:
Wydział, kierunek, semestr, grupa: WARiE, AiR, sem. 5, A3-L6	Imię i Nazwisko: 1. Filip Szulczyński 144517 2.Maksymilian Jaruga 144423	Punkty:
Data wykonania ćwiczenia: 25.01.2022		

Zawartość

SPIS ILUSTRACJI

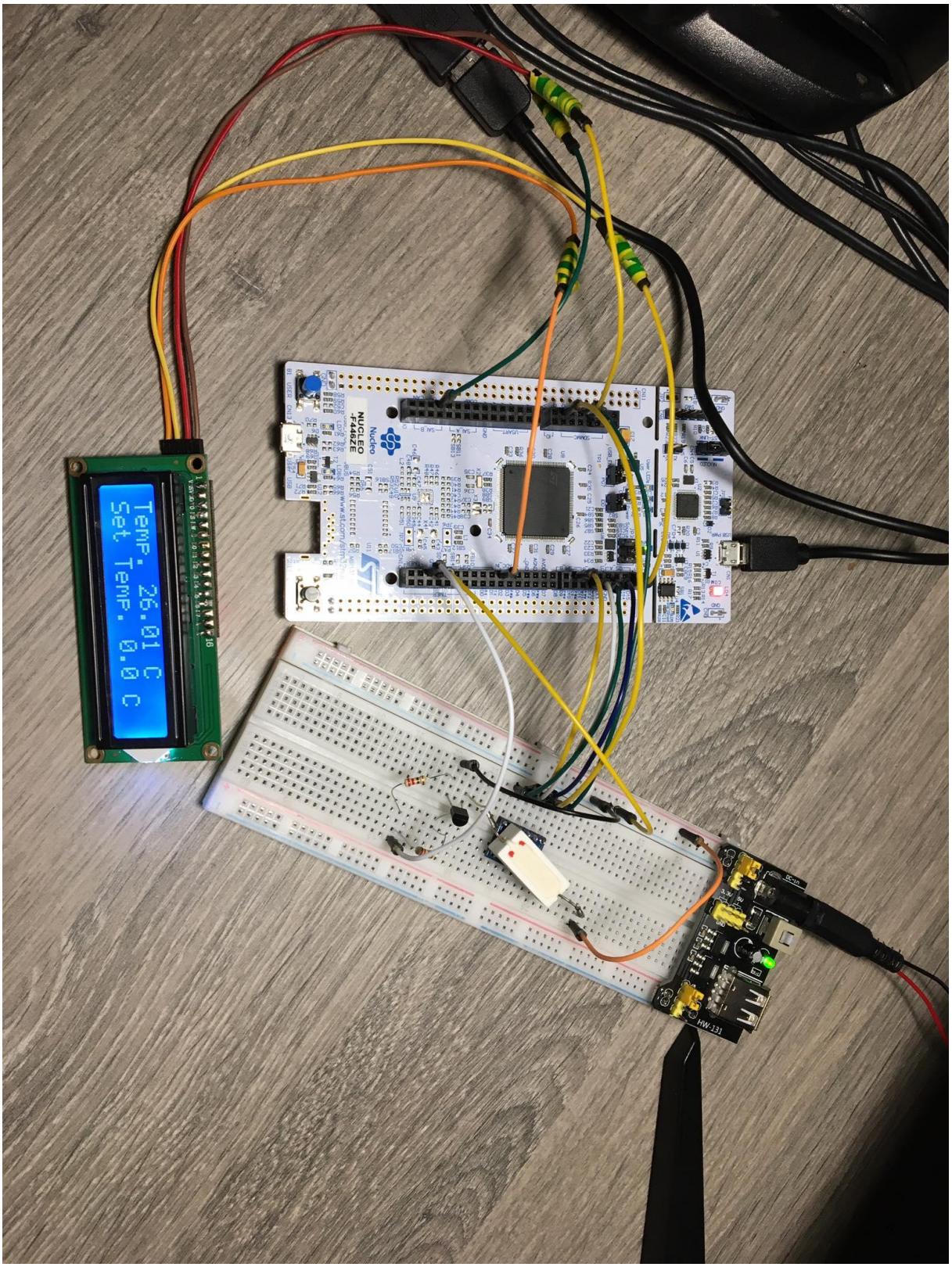
WSTĘP

Jako projekt końcowy z przedmiotu Systemy mikroprocesorowe wykonaliśmy zasugerowany przez prowadzącego układ grzewczy oparty na regulatorze o strukturze PID. Elementem grzewczym jest ceramiczny rezystor 5W o rezystancji 47Ω. Sterowanie układu zostało wykonane za pomocą tranzystora bipolarnego NPN. Pomiary temperatury elementu wykonawczego zrealizowany został w oparciu o sensor BMP280 podłączony do NUCLEO poprzez interfejs SPI. Wgląd w temperaturę referencyjną jak i rzeczywistą zrealizowany został na wyświetlaczu LCD 2x16 z konwerterem I2C. Poza wyświetlaczem, temperaturę aktualną możemy obserwować w czasie rzeczywistym za pomocą programu TelemetryViewer, na dodatek zostaje tam wyświetlony również sygnał sterujący w całym zakresie pracy układu.

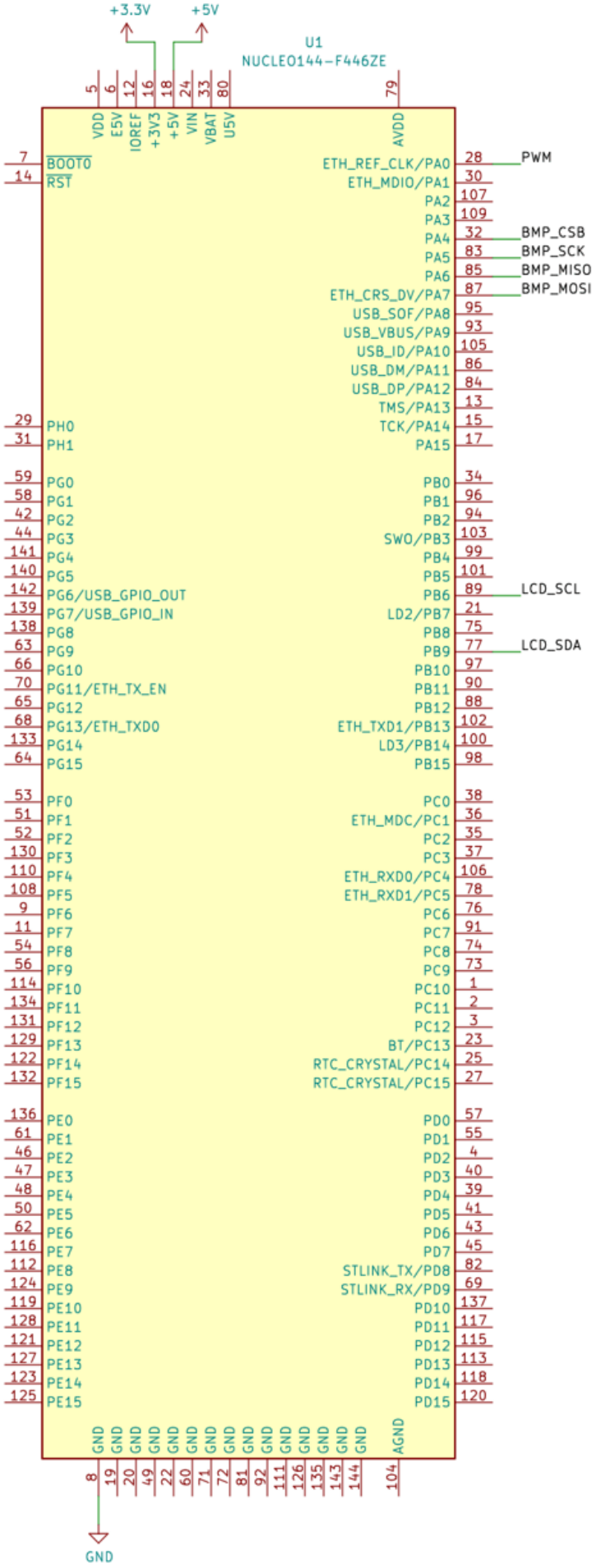
KOMPONENTY WYKORZYSTANE W PROJEKCIE

- Czujnik BMP280
- NUCLEO-F446ZE
- Wyświetlacz LCD HD44780 2x16 z konwerterem I2C PCF8574A
- Rezystor ceramiczny 5W o rezystancji 47Ω
- 2 rezystory 220Ω
- Tranzystor NPN BC237B
- Płytka zasilająca 12V → 5V
- Płytka stykowa
- Przewody

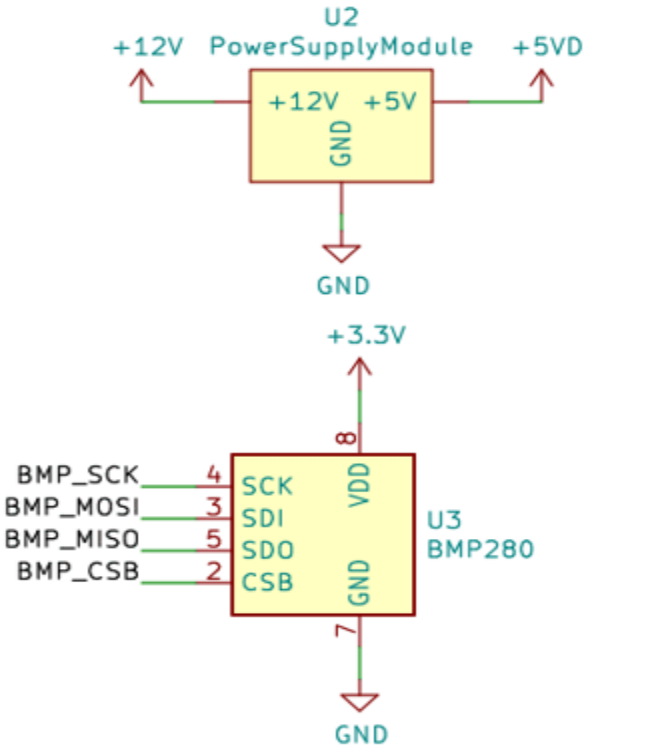
STWORZONY UKŁAD ORAZ SCHEMAT POŁĄCZEŃ.



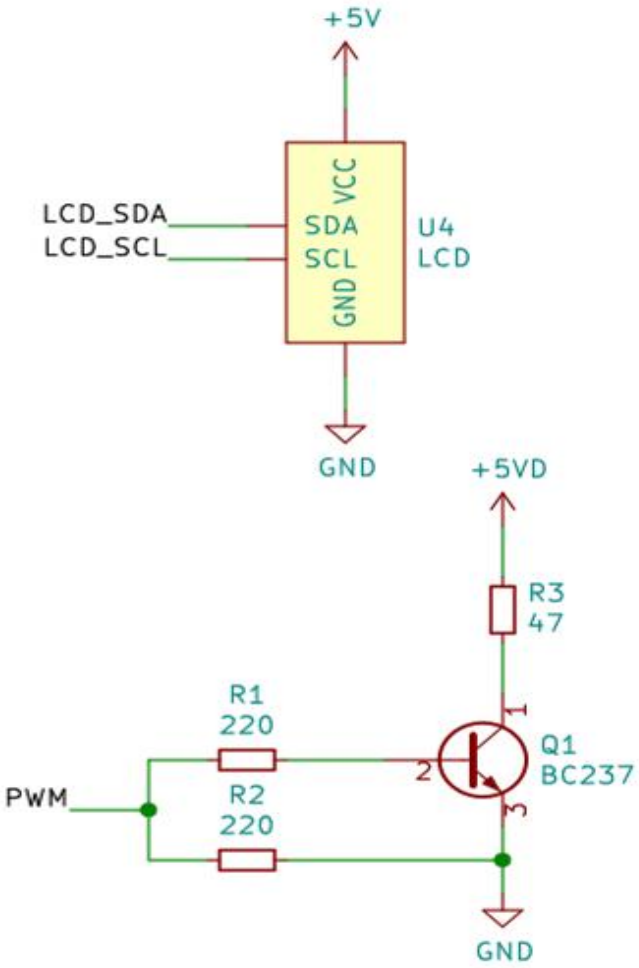
Rysunek 1 Zdjęcie wykonanego układu



Rysunek 2 Schemat- NUCLEO-F446ZE



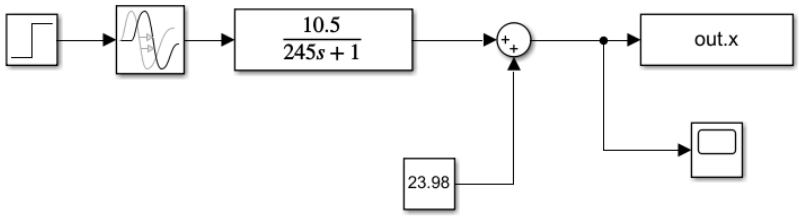
Rysunek 3 Schemat- moduł zasilania oraz czujnik temperatury



Rysunek 4 Schemat- uproszczone połączenie z LCD oraz część sterowana przez PWM

STWORZENIE I ZASYMULOWANIE OBIEKTU W ŚRODOWISKU MATLAB.

Chcąc wykonać i zasymulować obiekt za pomocą środowiska MATLAB, aby dobrać nastawy do regulatora PID, należało dokonać serii pomiarów przy stałym wypełnieniu PWM podanym na bramkę tranzystora NPN co powodowało otwarcie bramki i przepływ prądu przez rezystor ceramiczny a w konsekwencji grzanie. W naszym przypadku, dokonaliśmy pomiarów przy wypełnieniu równym 95%. Jak widać po samym przebiegu pomiarów, jest to obiekt inercyjny pierwszego rzędu z opóźnieniem transportowym. Po zbudowaniu w Simulinku obiektu, doświadczalnie wyznaczone zostały jego wartości poprzez sprawdzanie wyznaczonego modelu wraz z wykonanymi pomiarami.

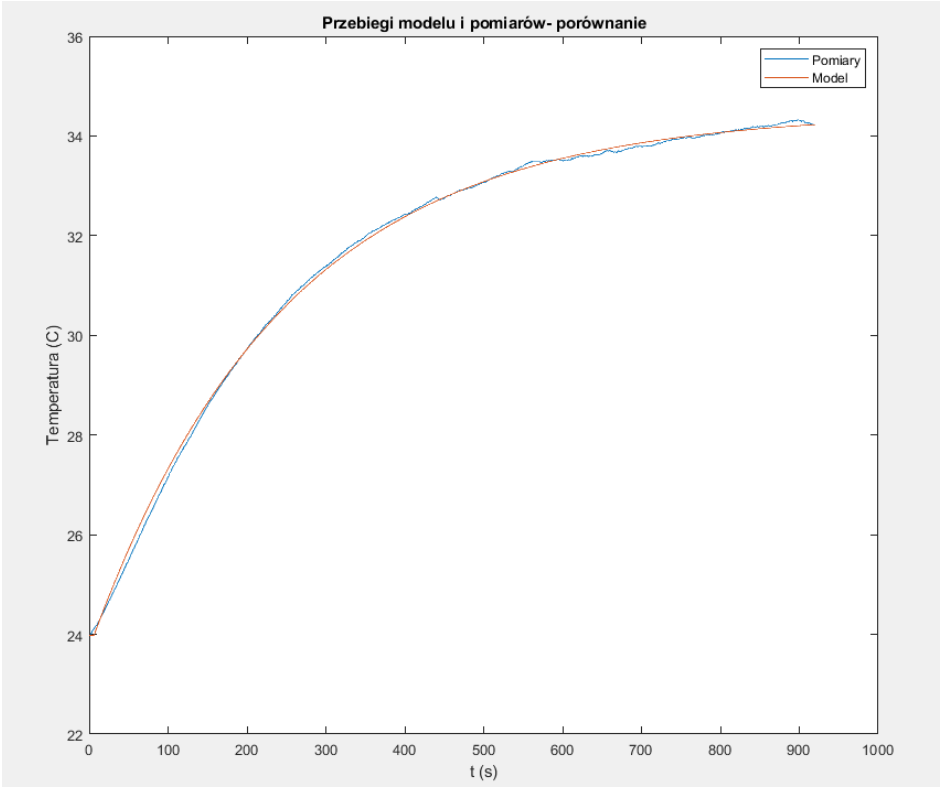


Rysunek 5 Model obiektu

Model miał więc następujące parametry:

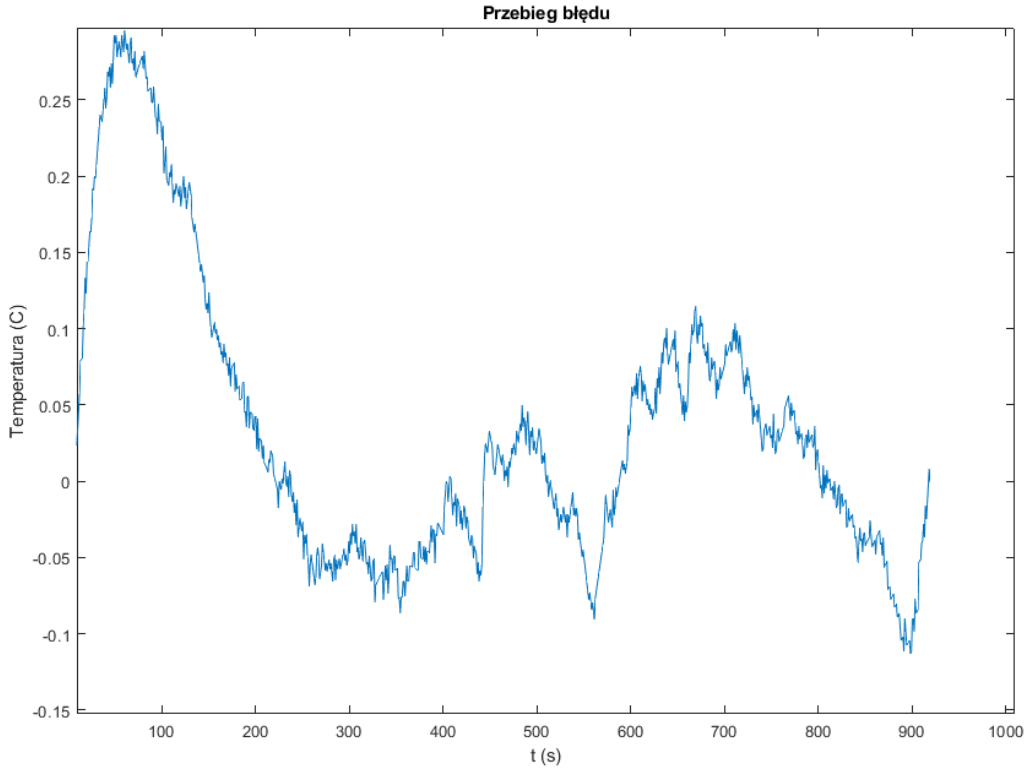
- K = 10,5
- T = 245 [s]
- T_d = 4 [s]

gdzie K jest wzmocnieniem obiektu, T stałą czasową, T_d stałą czasową opóźnienia transportowego.



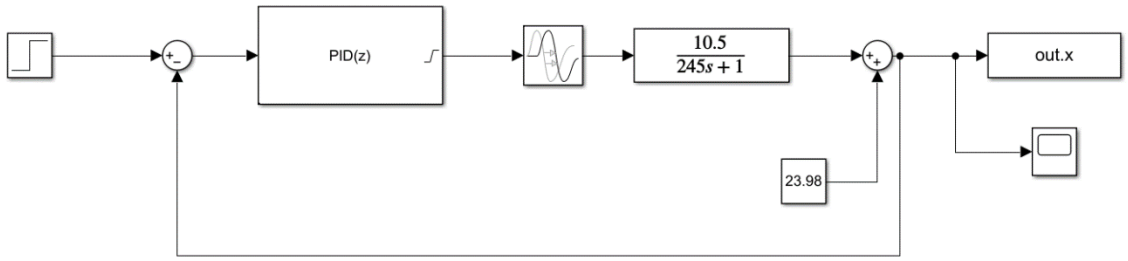
Wykres 1 Przebiegi modelu i pomiarów- porównanie

Wykres błędu między pomiarami a modelem wygląda następująco:



Wykres 2 Przebieg błędu w czasie

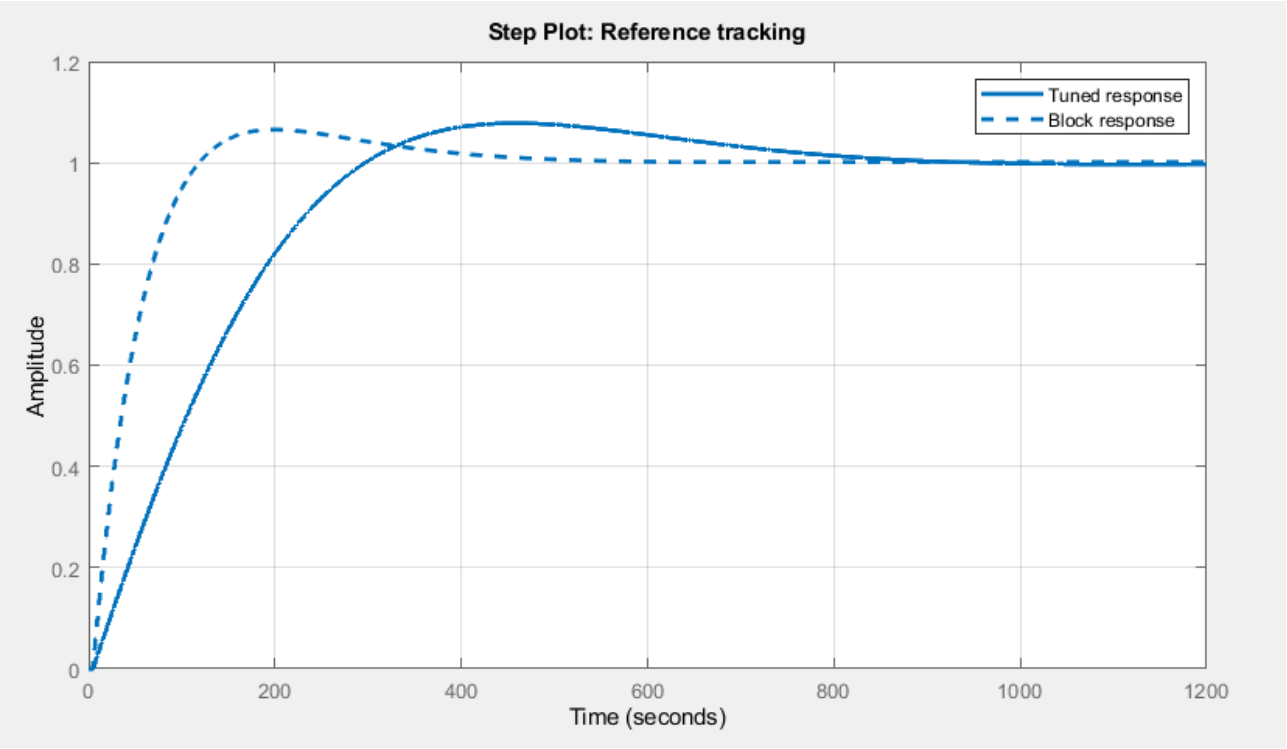
Następnie, do zbudowanego modelu, dołączony został regulator PID oraz zamknięta została pętla sprzężenia zwrotnego.



Rysunek 6 Model obiektu wraz z regulatorem PID

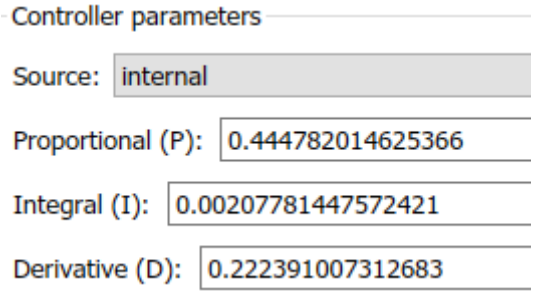
DOBÓR NASTAW REGULATORA PID.

Aby uzyskać nastawy dla regulatora, w Simulinku dołączony blok PID został ustawiony w tryb dyskretny. Do efektywniejszego strojenia wykorzystaliśmy funkcje autotune, która sama wyznaczyła wartości regulatora.



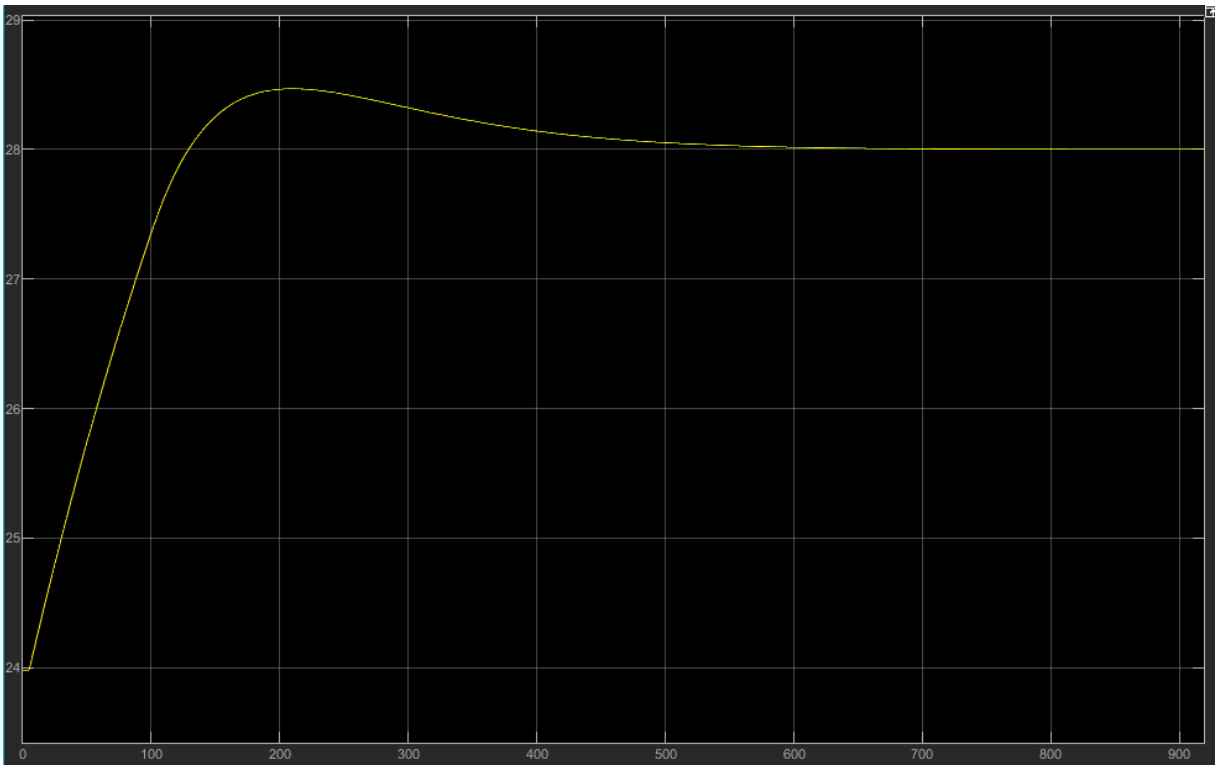
Wykres 3 PID autotune

Po przeanalizowaniu otrzymanego wyniku za pomocą funkcji autotune (wykres linii ciągłej) postanowiliśmy dokonać korekty parametrów za pomocą suwaków i otrzymaliśmy niżej przedstawione nastawy dla wykresu linii przerywanej.



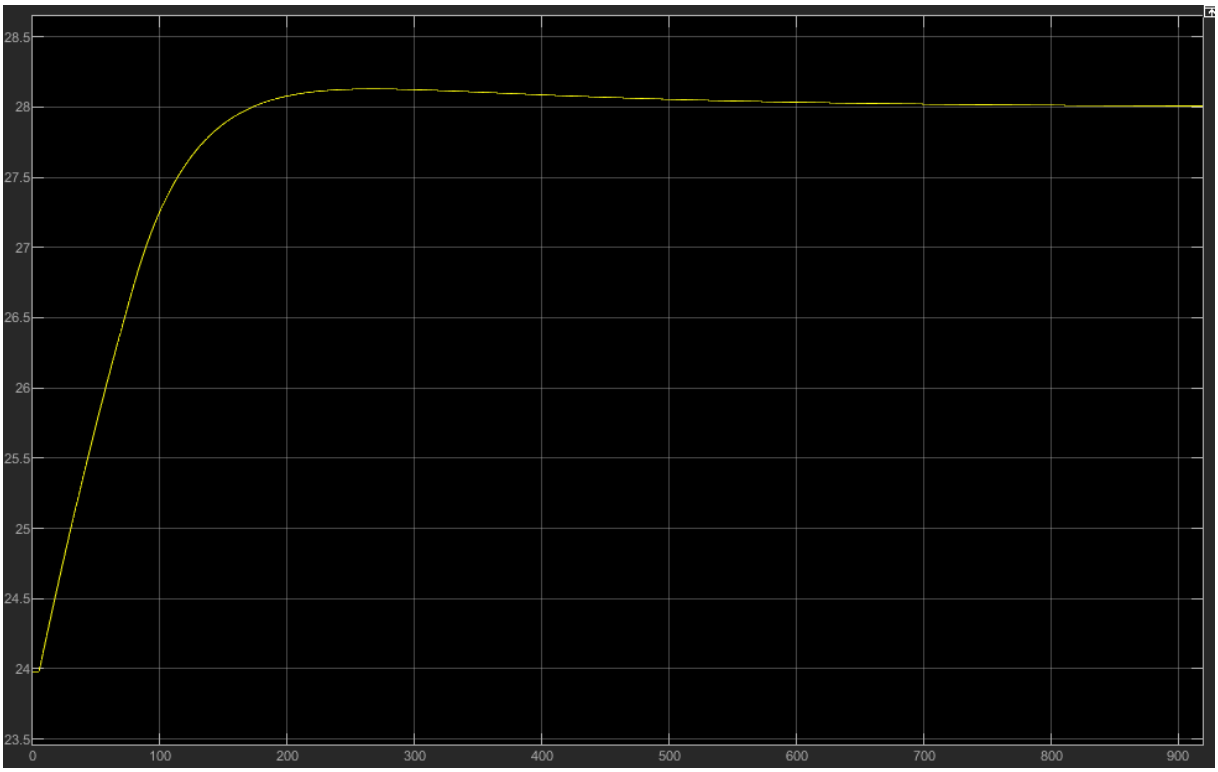
Rysunek 7 Wyznaczone parametry PID

Dzięki dołączeniu funkcji „Scope” w Simulinku, można było zobaczyć spodziewane efekty działania naszego regulatora.



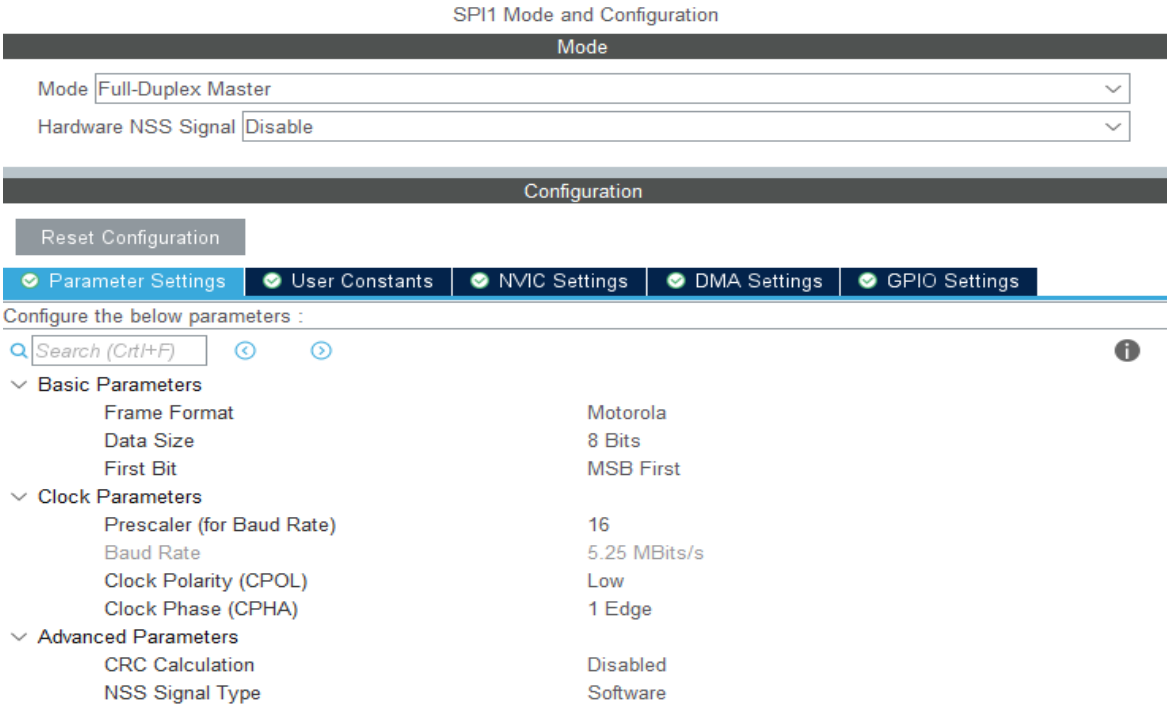
Wykres 4 PID dla zadanej temperatury 28 ^{o}C

Aby poprawić nieco poziom przeregulowania, dołączony został anti-windup.

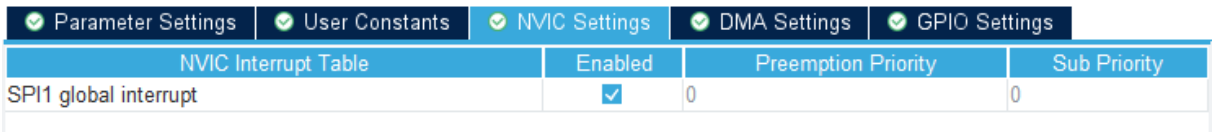


Wykres 5 PID dla zadanej temperatury 28 ^{o}C z anti-windupem

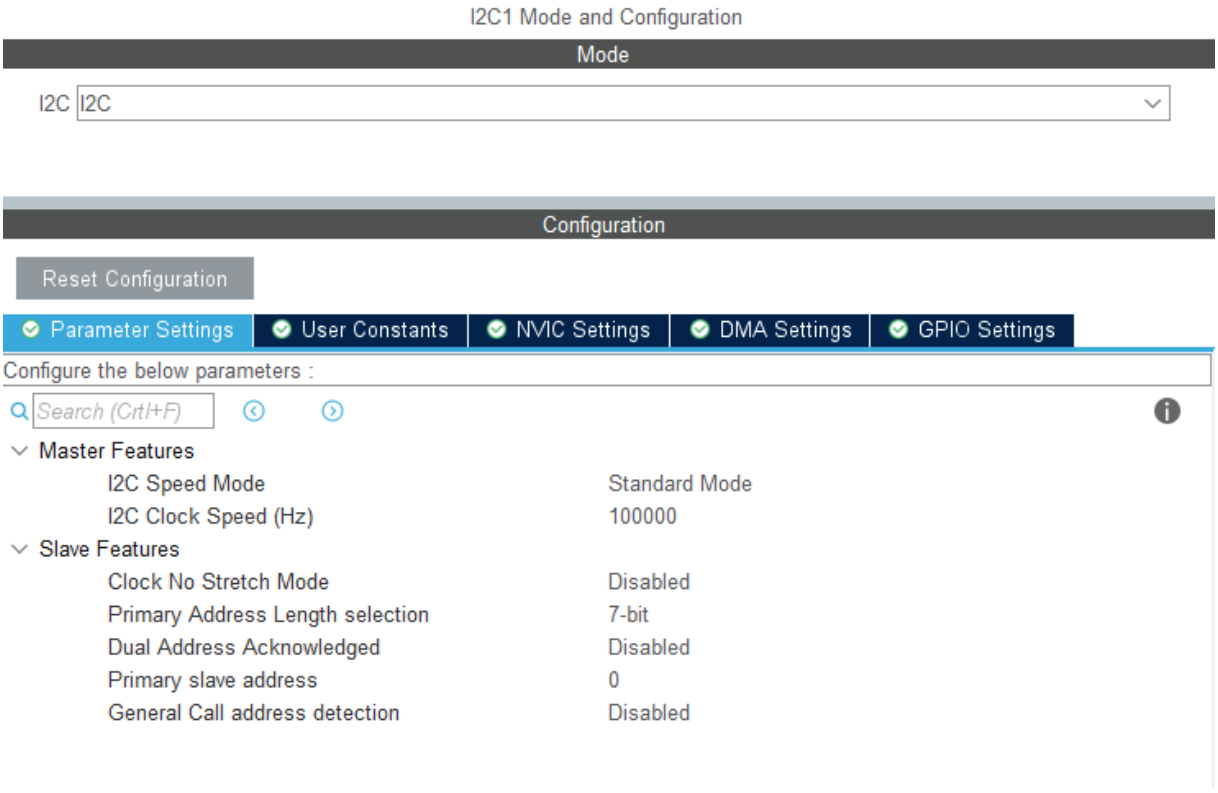
KONFIGURACJA NUCLEO-F446ZE W STM32CUBEIDE.



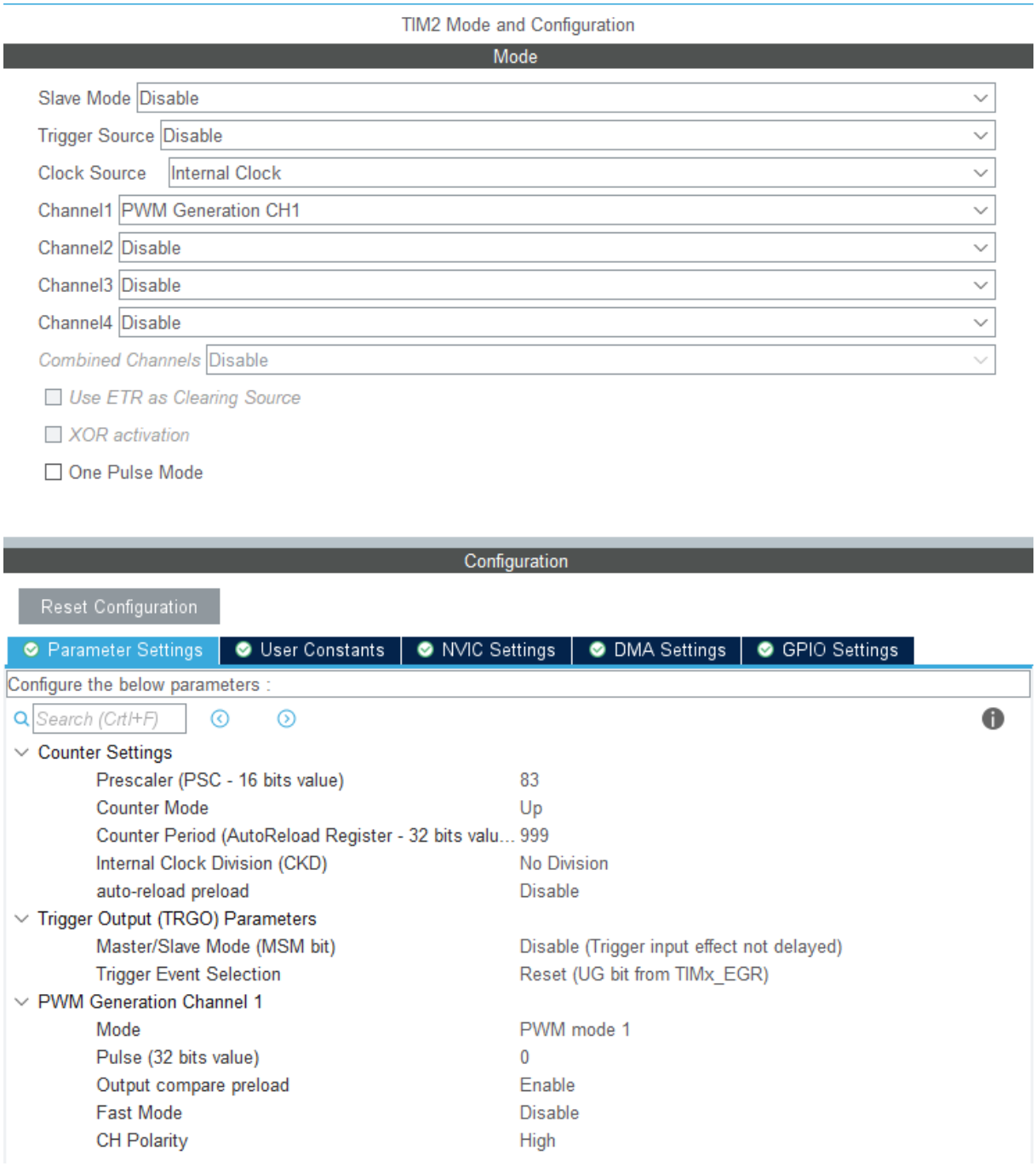
Rysunek 8 Konfiguracja SPI



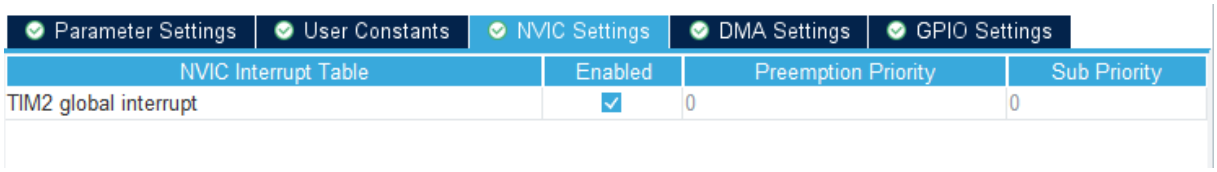
Rysunek 9 Włączenie przerwań dla SPI



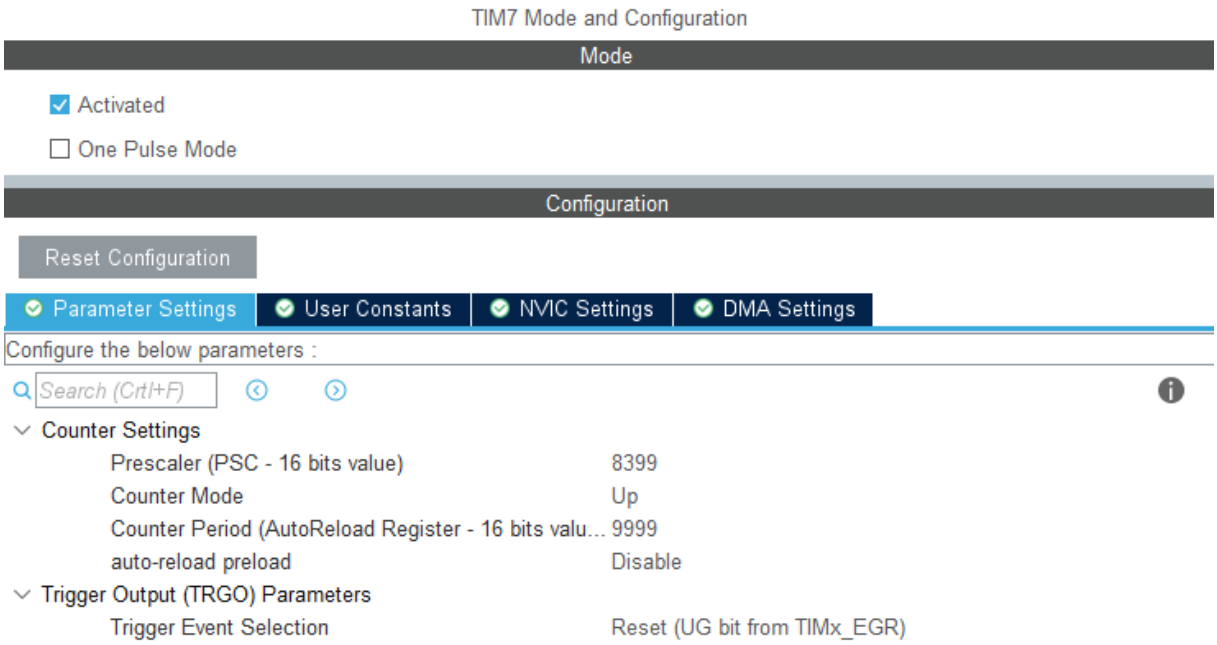
Rysunek 10 Konfiguracja I2C



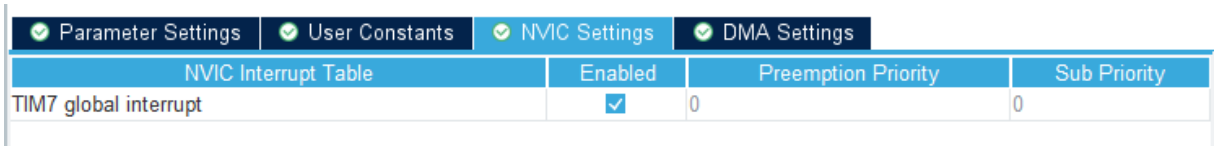
Rysunek 11 Konfiguracja TIM2



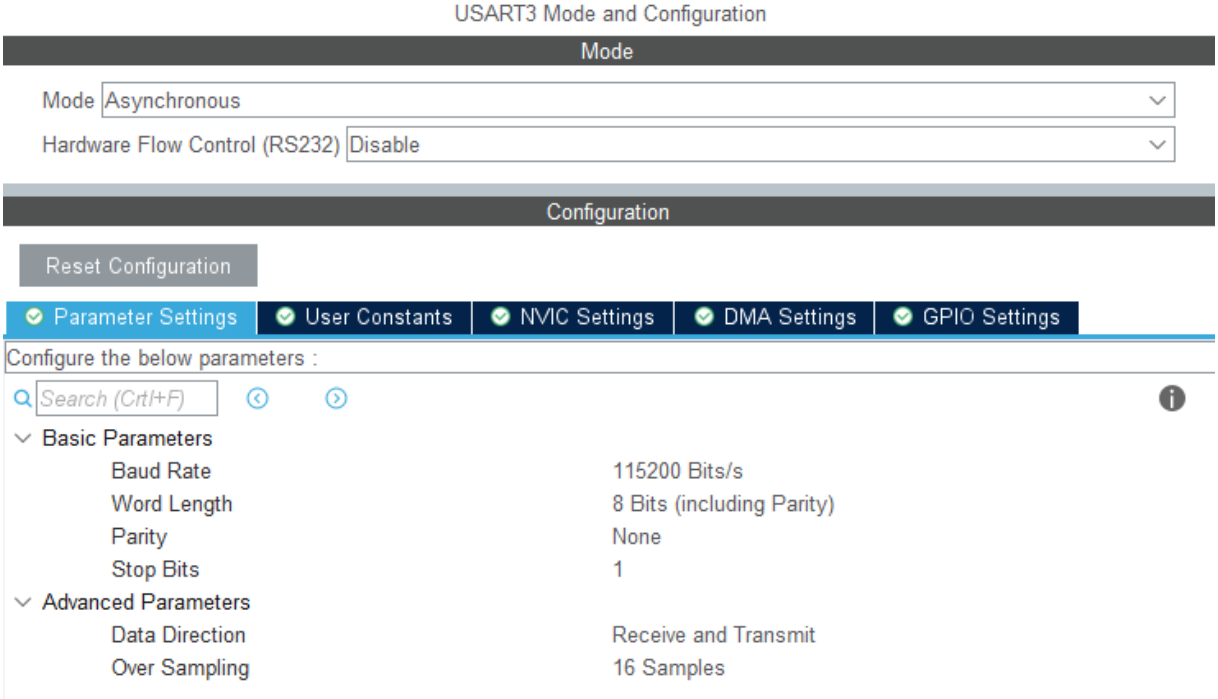
Rysunek 12 Włączenie przerwań dla TIM2



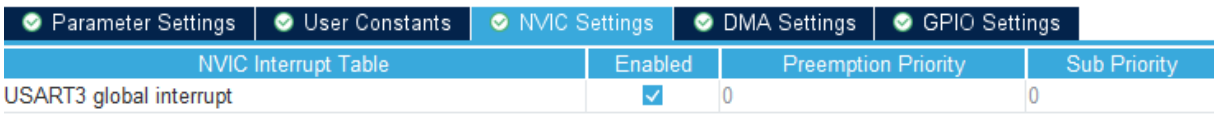
Rysunek 13 Konfiguracja TIM7



Rysunek 14 Włączenie przerwań dla TIM7



Rysunek 15 Konfiguracja USART3



Rysunek 16 Włączenie przerwań dla USART3

KOD WGRANY DO MIKROPROCESORA, Z REALIZACJĄ REGULATORA PID ORAZ SYSTEMU ANTI-WINDUP

```
/* Private includes ----------------
/* USER CODE BEGIN Includes */
#include "BMPXX80.h"
#include "i2c-lcd.h"
#include "string.h"
#include "stdio.h"
/* USER CODE END Includes */
```

Rysunek 17 Kod- załączone biblioteki

```
/* USER CODE BEGIN PV */
float temperature, pressure, setvalue, duty_f, u;
char text[20];
char lcdtext[20];
char receive[5];

uint16_t duty;

typedef struct{
        float Kp;
        float Ki;
        float Kd;
        float dt;
}pid_parameters;

typedef struct{
    pid_parameters p;
    float previous_err;
    float previous_int;
    float windup;
}pid;

pid pid1 = {.p.Kp = 0.445, .p.Ki = 0.002, .p.Kd = 0.222, .p.dt = 1.0, .previous_err = 0, .previous_int = 0};
/* USER CODE END PV */
```

Rysunek 18 Kod- utworzone zmienne i struktury

```
/* USER CODE BEGIN 2 */
BMP280_Init(&hspi1, BMP280_TEMPERATURE_16BIT, BMP280_STANDARD, BMP280_FORCEDMODE);
lcd_init();
HAL_TIM_Base_Start_IT(&htim7);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_UART_Receive_IT(&huart3, (uint8_t*)receive, 4);
/* USER CODE END 2 */
```

Rysunek 19 Kod- inicjalizacje

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    lcd_clear ();
    lcd_put_cur(0, 0);
    sprintf((char*)text, "Temp. %.2f C", temperature);
    lcd_send_string(text);

    lcd_put_cur(1, 0);
    sprintf((char*)text, "Set Temp. %.1f C",setvalue);
    lcd_send_string(text);

    HAL_Delay(1000);


  /* USER CODE END WHILE */
```

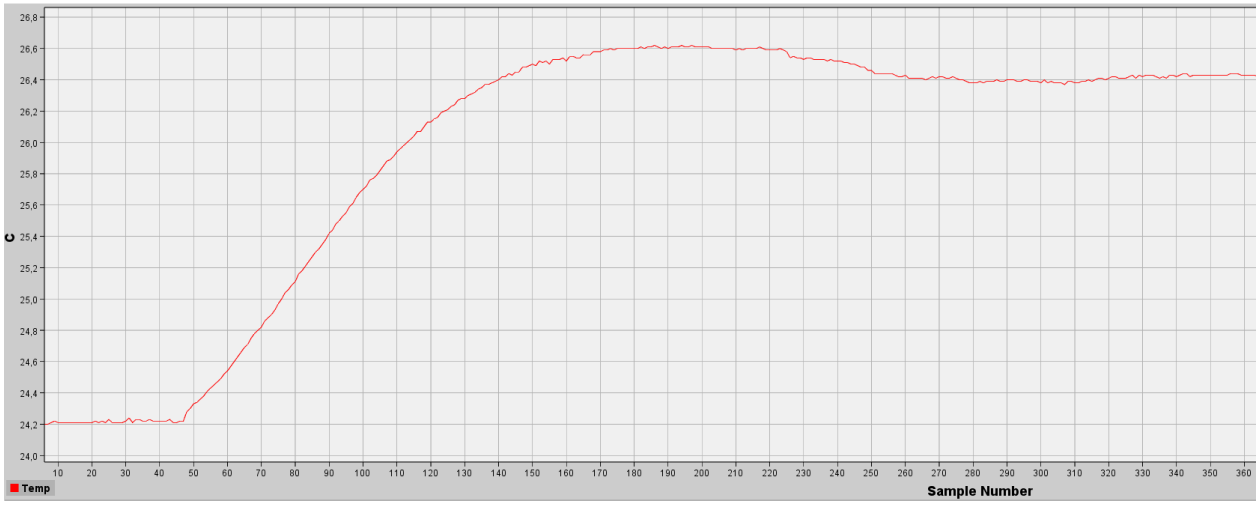
Rysunek 20 Kod- pętla while(1)

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart)
{
    sscanf((char*)receive,"%f",&setvalue);
    HAL_UART_Receive_IT(&huart3, (uint8_t*)receive, 5);
}



float PID(pid* pid, float setvalue, float temperature){
    float u = 0, P, I, D, error = 0, integral = 0, derivative = 0;

    if(setvalue != 0){
    error = setvalue - temperature;

    P = pid->p.Kp * error;

    integral = pid->previous_int + (error +(3*pid->windup)+ pid->previous_err);
    pid->previous_int = integral;
    I = pid->p.Ki * integral * (pid->p.dt/2.0);

    derivative = (error - pid->previous_err)/pid->p.dt;
    pid->previous_err = error;
    D = pid->p.Kd * derivative;

    u = P + I + D;
    }
    return u;
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
    if(htim->Instance == TIM7){
              duty_f = 0;
              BMP280_ReadTemperatureAndPressure(&temperature, &pressure);
              u=PID(&pid1, setvalue, temperature);
              if(u > 1){
                    pid1.windup = 1.0 - u;
                    duty_f = 999.0;
                }
                else if(u < 0){
                    pid1.windup = 0.0 - u;
                    duty_f = 0;
                }
                else{
                    pid1.windup = 0.0;
                    duty_f = 999 * u;
                }
              duty = (uint16_t) duty_f;
              __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
              sprintf((char*)text, "%.2f, %u, \n\r", temperature, duty);
              //sprintf((char*)text, "%.2f, ", temperature);
              HAL_UART_Transmit_IT(&huart3, (uint8_t*)text, strlen(text));

    }
}
/* USER CODE END 0 */
```

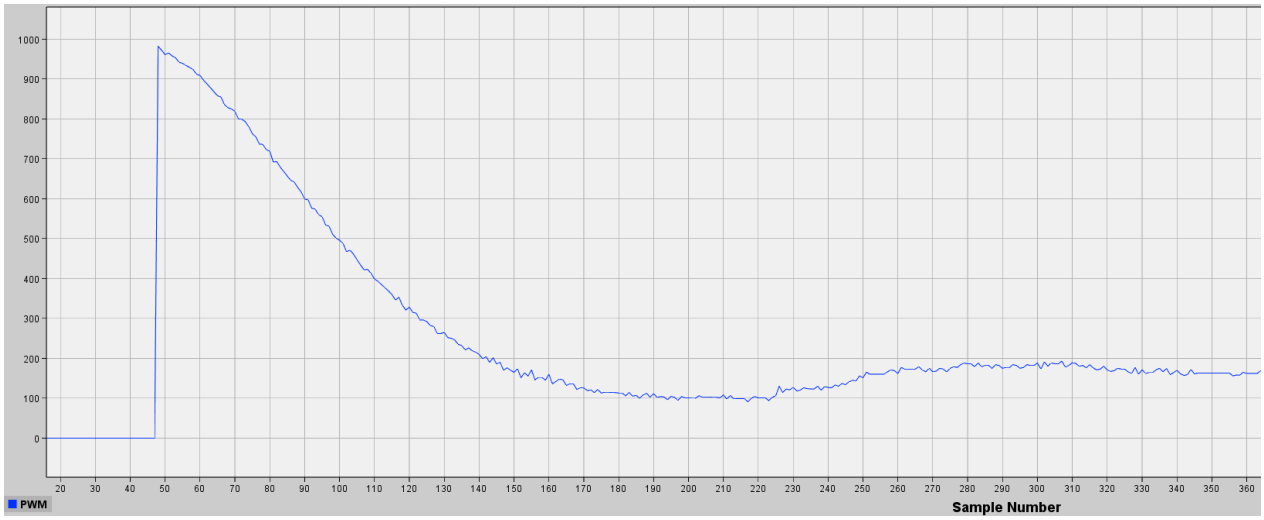
Rysunek 21 Kod- utworzone funkcje

DEMONSTRACJA PRZEBIEGÓW OTRZYMANYCH ZA POMOCĄ PROGRAMU TELEMETRYVIEWER

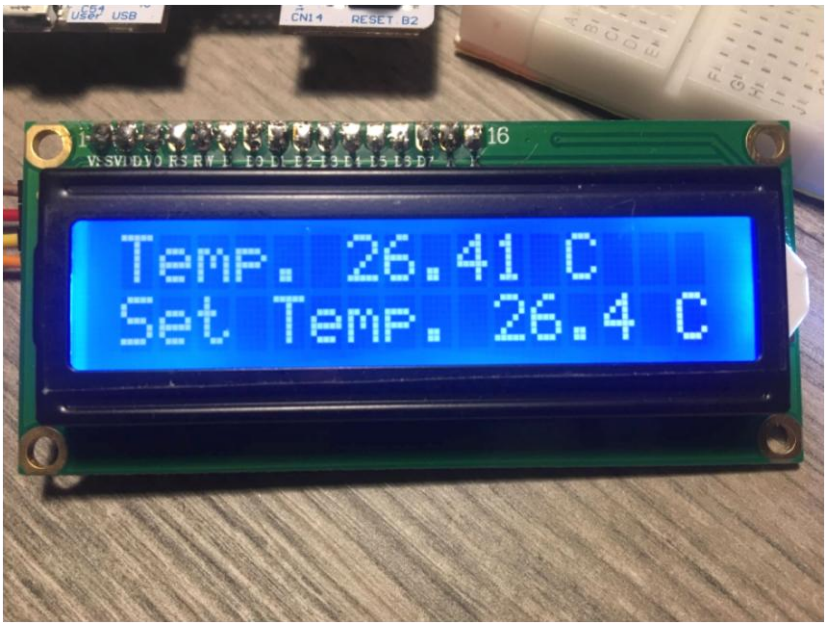
Przebiegi dla temperatury zadanej 26.4°C (grzanie)



Wykres 6 Przebieg temperatury- zadane 26.4°C (grzanie)



Wykres 7 Przebieg PWM- zadane 26.4°C (grzanie)

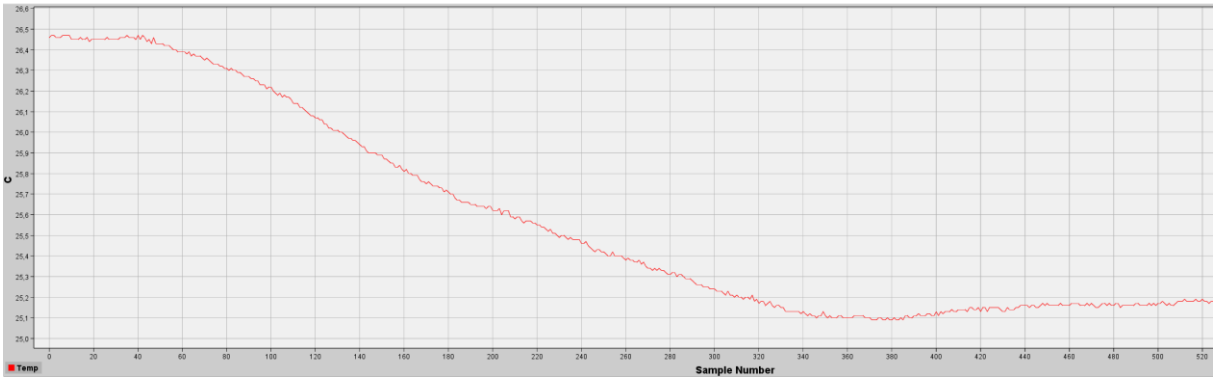


Rysunek 22 26.4°C- wyświetlacz LCD

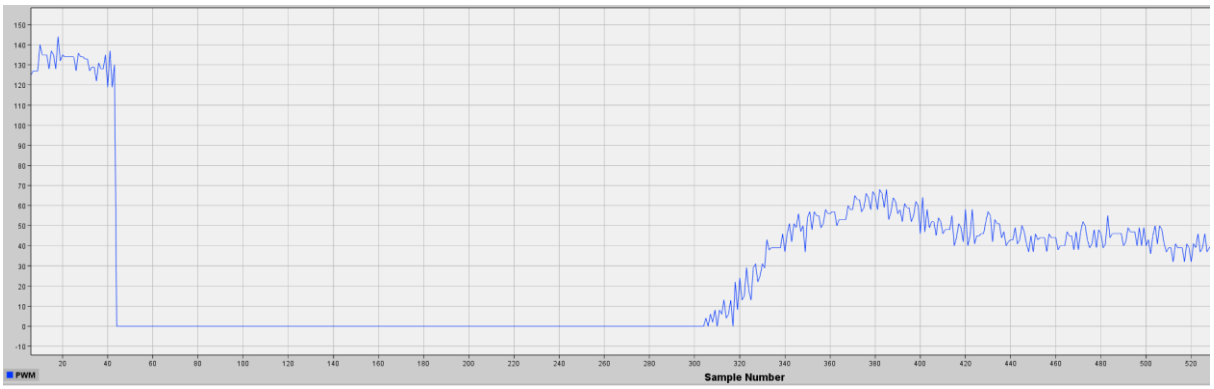
W stanie ustalonym wartość temperatury zmieniała się między 26.40°C, a wartością 26.43°C. Zatem uchyb ustalony wynosi:

$$e_{ust} = \frac{|26.4 - 26.43|}{26.4} * 100\% = \frac{0.03}{26.4} * 100\% = 0.11\%$$

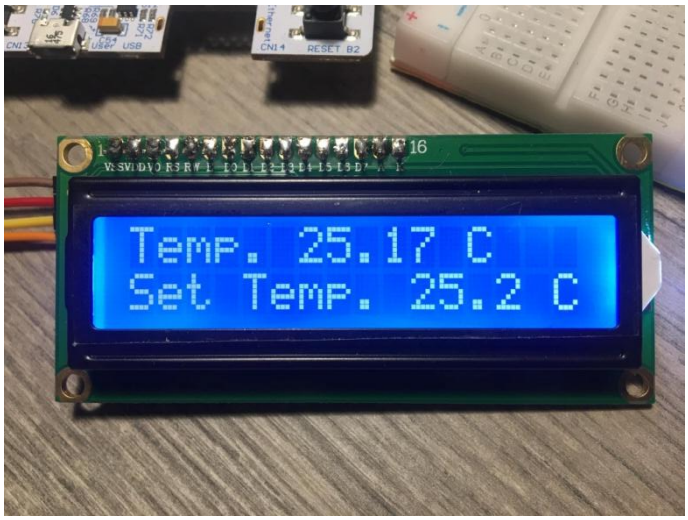
Przebiegi dla temperatury zadanej 25.2° C (chłodzenie)



Wykres 8 Przebieg temperatury- zadane 25.2°C (chłodzenie)



Wykres 9 Przebieg PWM zadane 25.2°C (chłodzenie)

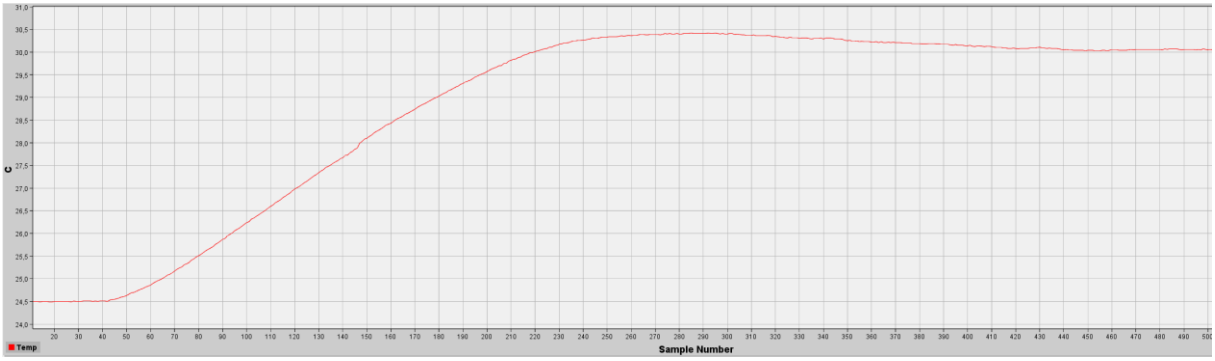


Rysunek 23 25.2°C- wyświetlacz LCD

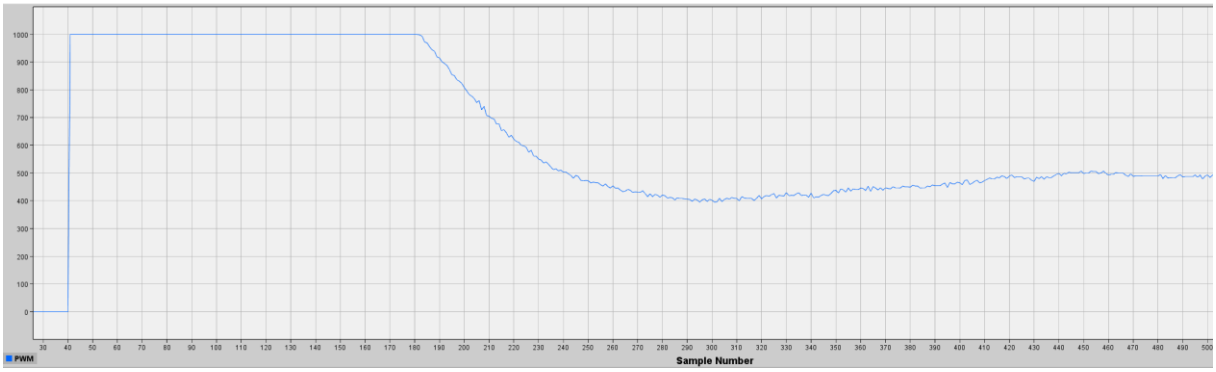
W stanie ustalonym wartość temperatury zmieniała się między 25.16°C, a wartością 25.2°C. Zatem uchyb ustalony wynosi:

$$e_{ust} = \frac{25.2 - 25.16}{25.2} * 100\% = \frac{0.04}{25.2} * 100\% = 0.16\%$$

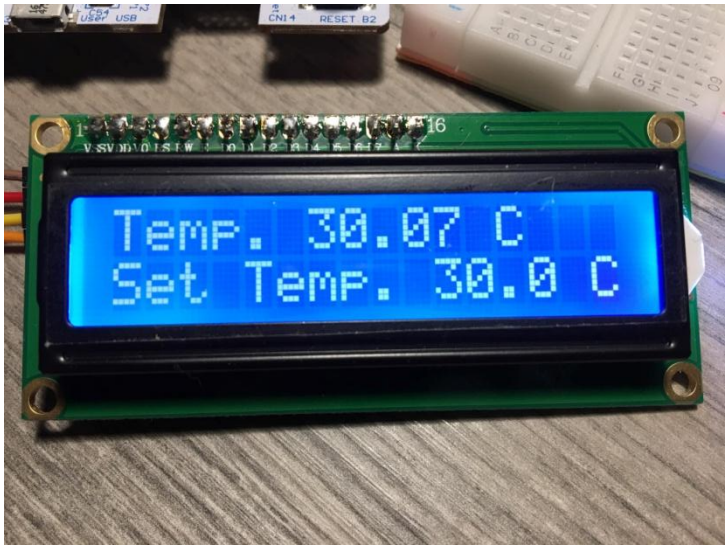
Przebiegi dla temperatury zadanej 30° C (grzanie)



Wykres 10 Przebieg temperatury- zadane 30°C (grzanie)



Wykres 11 Przebieg PWM- zadane 30°C (grzanie)

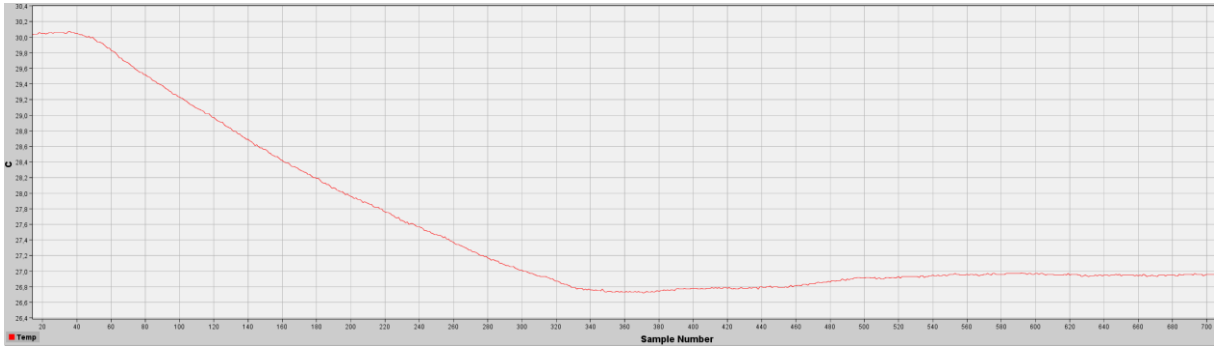


Rysunek 24 30°C- wyświetlacz LCD

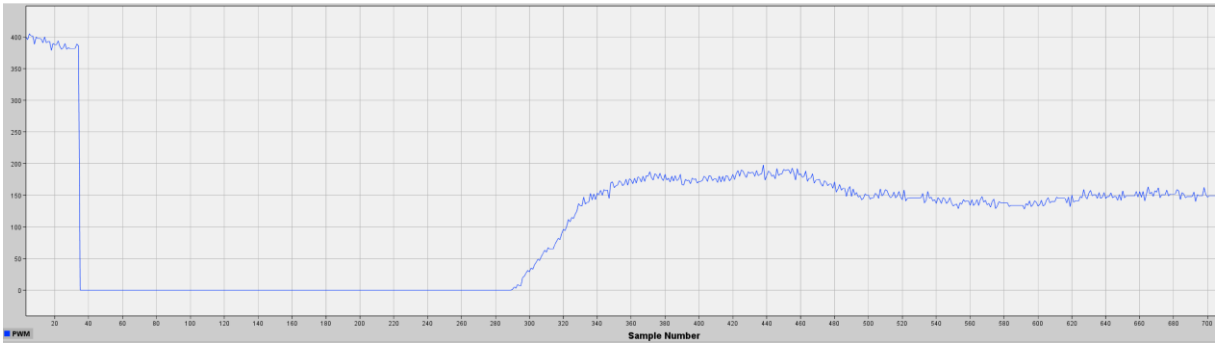
W stanie ustalonym wartość temperatury zmieniała się między 30.08°C, a wartością 30.0°C. Zatem uchyb ustalony wynosi:

$$e_{ust} = \frac{30 - 30.08}{30} * 100\% = \frac{0.08}{30} * 100\% = 0.27\%$$

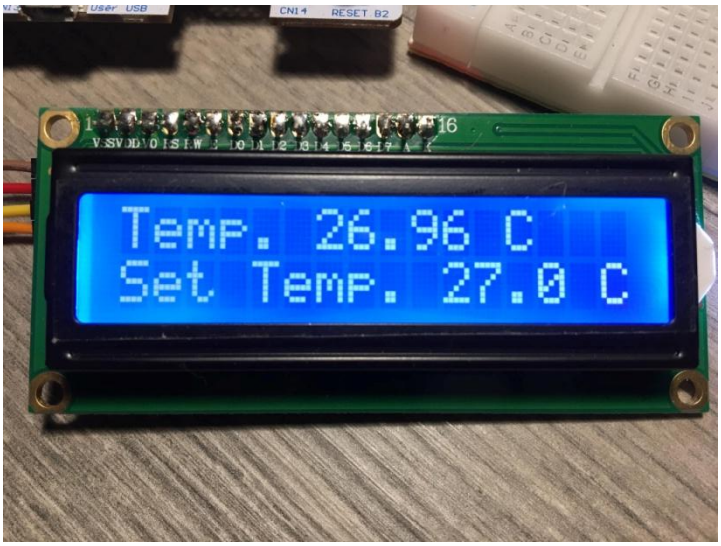
Przebiegi dla temperatury zadanej 27° C (chłodzenie)



Wykres 12 Przebieg temperatury- zadane 27°C (chłodzenie)



Wykres 13 Przebieg PWM zadane 27°C (chłodzenie)



Rysunek 25 27°C- wyświetlacz LCD

W stanie ustalonym wartość temperatury zmieniała się między 26.96°C, a wartością 27.0°C. Zatem uchyb ustalony wynosi:

$$e_{ust} = \frac{27 - 26.96}{27} * 100\% = \frac{0.04}{27} * 100\% = 0.15\%$$

SYSTEM KONTROLI WERSJI

Zaprezentowany projekt jest umieszczony na GitHubie:
https://github.com/Morgaliel/PID-TempRegulation