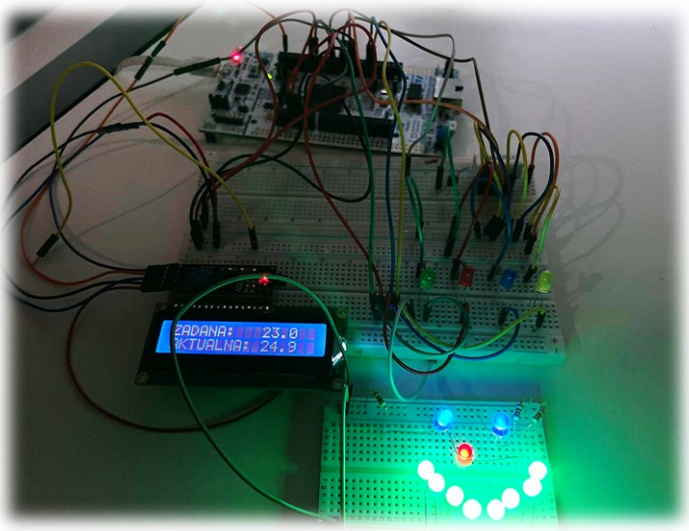
AUTOMATYCZNA REGULACJA TEMPERATURY

Mikroprocesorowy system sterowania i pomiaru. Regulator PID.



AUTORZY PROJEKTU	DOMINIKA FILIPIAK WOJCIECH FLORCZAK
GRUPA	GRUPA DZIEKAŃSKA – A1 GRUPA LABORATORYJNA L2
POLITECHNIKA POZNAŃSKA WYDZIAŁ WARIE AUTOMATYKA I ROBOTYKA	

Spis treści

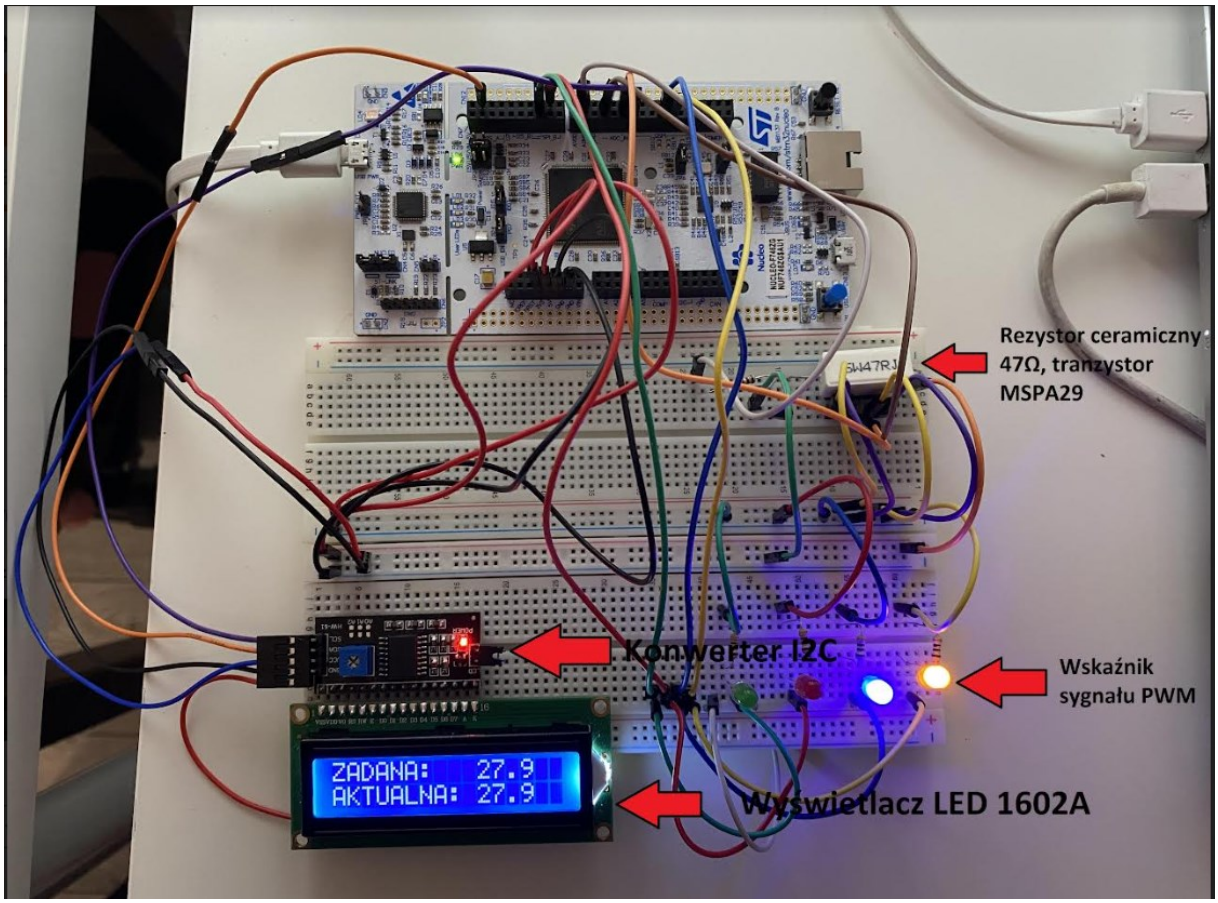
1. Założenia projektowe

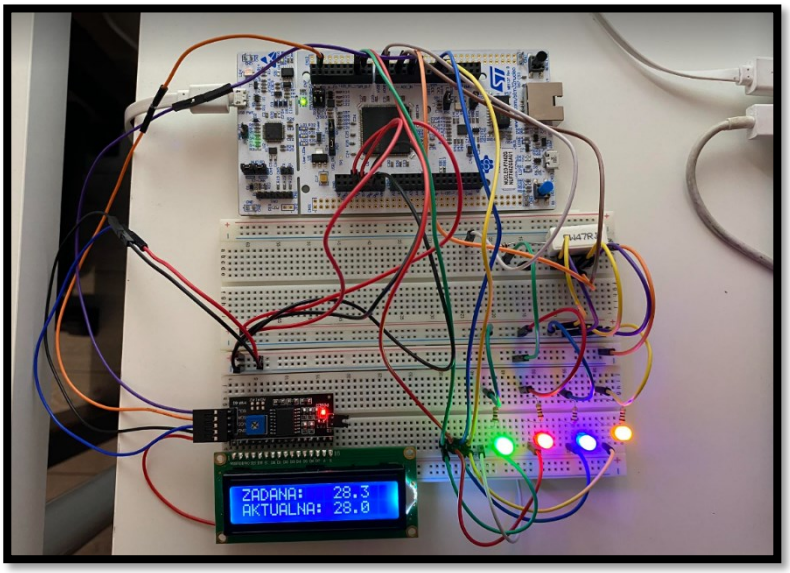
Zadaniem projektowym było stworzenie układu automatycznej regulacji temperatury rezystora sterowanego za pomocą tranzystora.

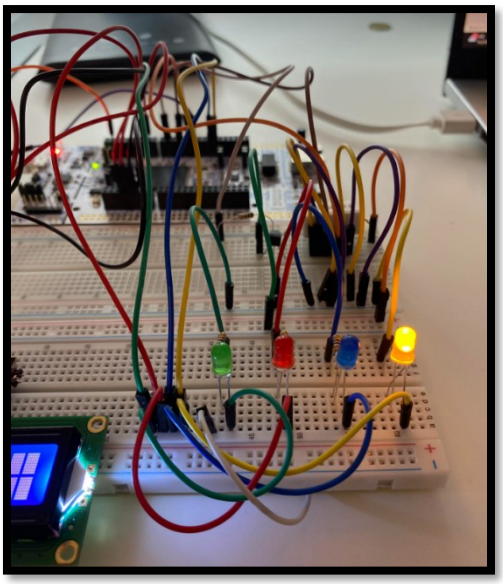
Wykorzystane komponenty:

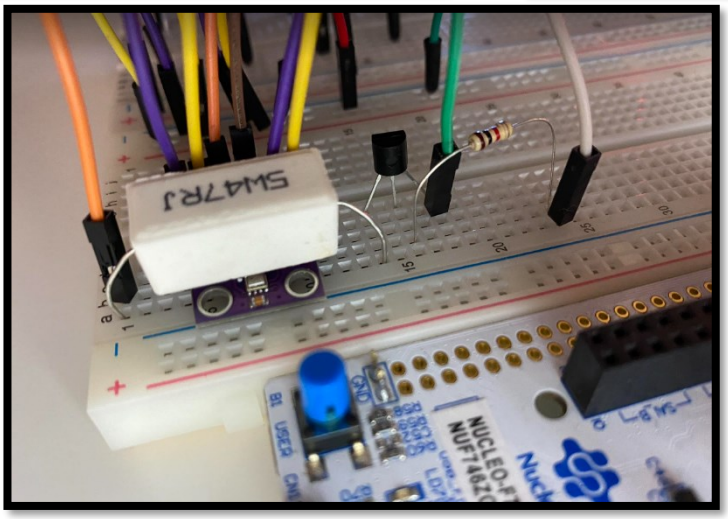
- Nucleo F746ZG
- Rezystor ceramiczny 47Ω
- Tranzystor bipolarny NPN Darlington MPSA29
- Rezystor 220Ω
- Czujnik BMP280
- Wyświetlacz LCD 1602A z konwerterem I2C
- Przewody połączeniowe
- Diody LED (żółta, niebieska, czerwona, zielona)

2. Realizacja układu





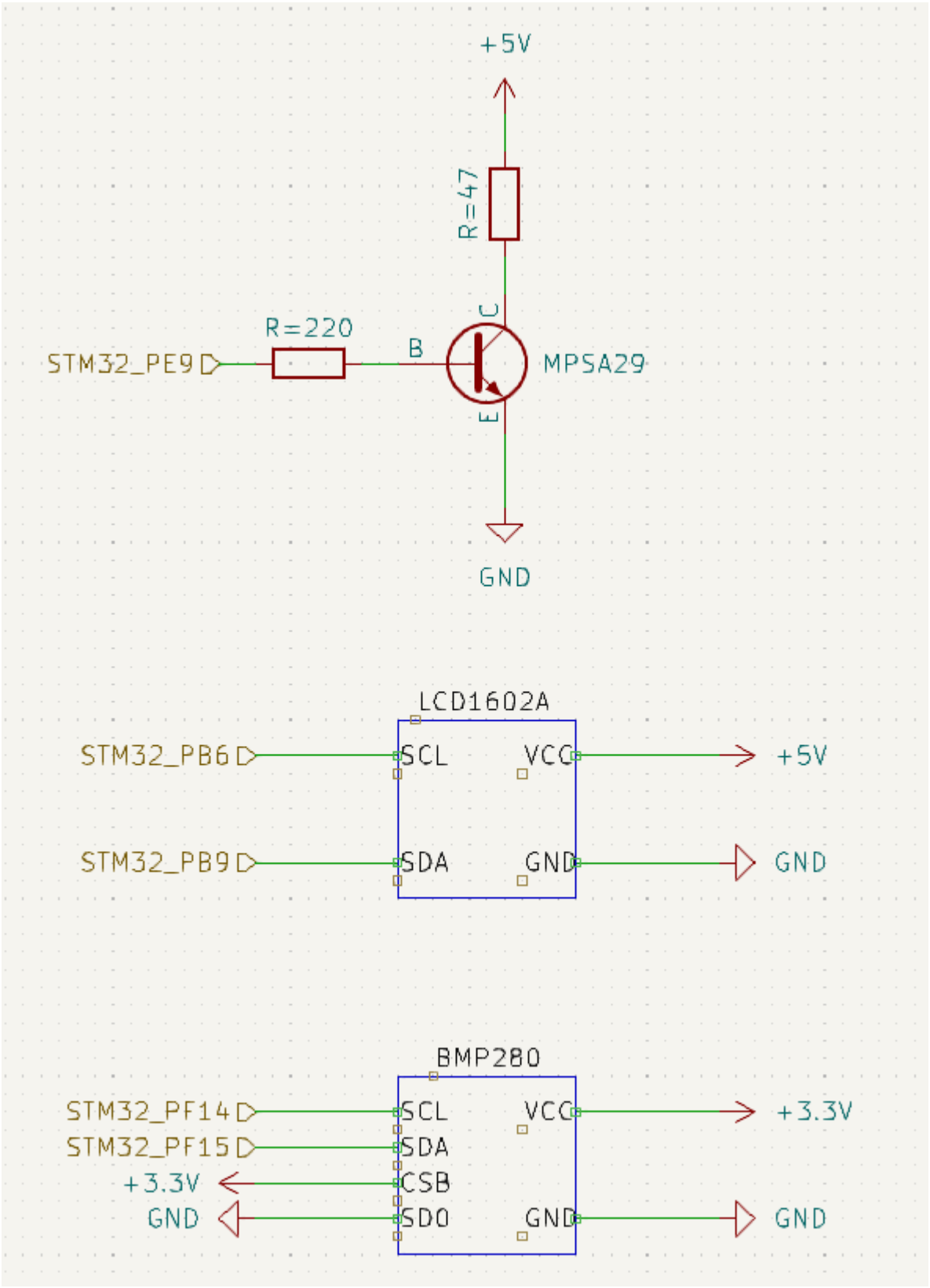




3. Analiza działania układu

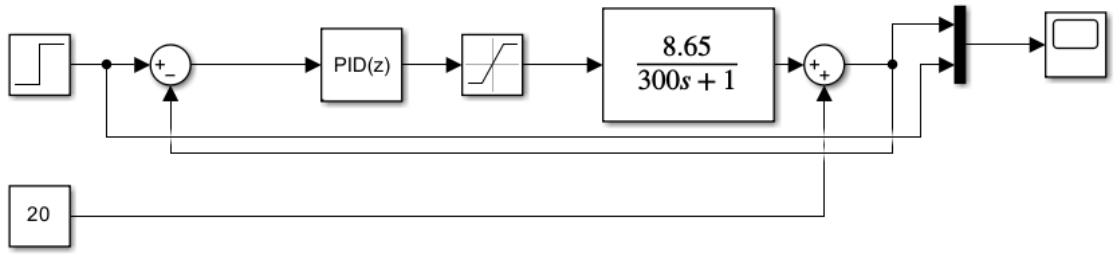
W projekcie zajmujemy się regulacją temperatury rezystora ceramicznego 47 [Ω], który został umieszczony tuż nad czujnikiem temperatury i ciśnienia BMP280. Przepływem prądu przez wspomniany rezystor steruje tranzystor NPN, do którego dobrano rezystor 220 [Ω]. Użyto również wyświetlacza LCD wraz z konwerterem I2C, który umożliwia proste przesyłanie danych. Na wspomnianym wyświetlaczu możliwy jest odczyt temperatury zadanej (jej zmiana jest możliwa dzięki wbudowanemu w płytkę Nucleo przyciskowi USER z krokiem 0.1 [˚C], natomiast resetowanie tej wartości dokonujemy poprzez również wbudowany przycisk RESET) oraz aktualna temperatura odczytywana z czujnika BMP280. Układ zasilany jest bezpośrednio z płytki Nucleo – wykorzystujemy zarówno napięcie 3.3 [V], jak i 5 [V]. Algorytm działania zadania regulacji stałowartościowej realizowany jest na płytce NucleoF746ZG opartej na mikrokontrolerze STM32F746ZGT6. Dodatkowo stworzyliśmy wizualizację wypełnienia sygnału PWM (diody obok wyświetlacza). Dioda żółta migająca oznacza, że wartość sygnału PWM jest równa 0. Natomiast zapala się w momencie gdy wartość wypełnienia PWM oscyluje w zakresie <0 ; 25> [%]. Dioda niebieska informuje o wypełnieniu z zakresu <25 ; 50> [%]. I kolejno diody czerwona <50 ; 75> [%], zielona <75 ; 100 > [%].

4. Schemat elektroniczny

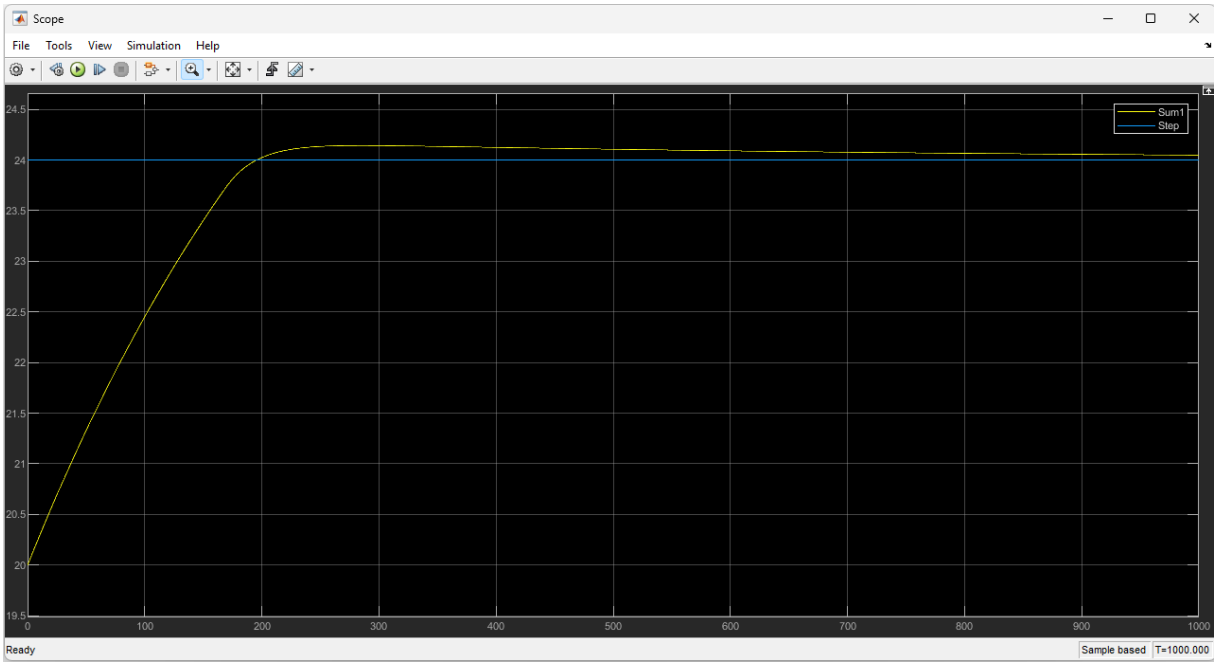


5. Model Matlab – Simulink

Do wykonania modelu posłużyliśmy się transmitancją rezystora, która została podana na jednym z wykładów. Zaprojektowaliśmy model URA z wykorzystaniem Matlab -Simulink.



Rys. 1 Schemat URA wykonany w Simulink



Rys. 2 Wykres przebiegu temperatury rezystora ceramicznego oraz wartości zadanej (tutaj na wartości 24)

Widzimy, że na powyższej symulacji występuje minimalne przeregulowanie rzędu 0,5%, jest więc ono pomijalnie małe. Jeśli chodzi o wartości uchybu ustalonego to również mieści się ona w zakresie tolerancji (>1%).

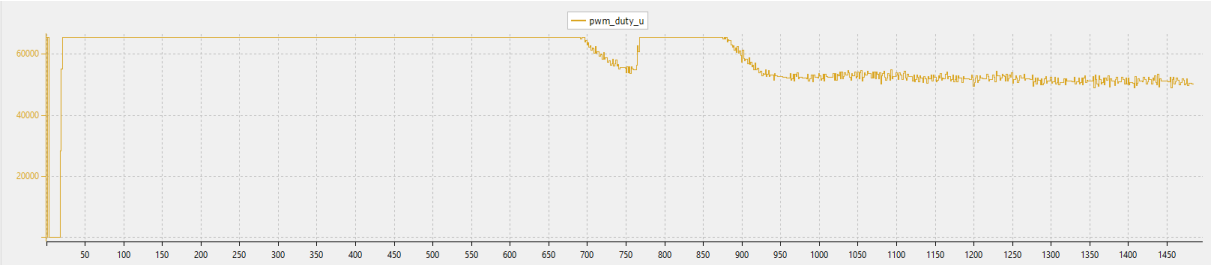
Doboru nastaw regulatora dokonano przy pomocy bloku Discrete PID Controller. Blok ten pomógł nam tak dobrać nastawy, aby spełnione były założenia projektowe, czyli odpowiednio niski uchyb ustalony. Wyznaczone nastawy przyjęły wartości:

$K_P = 1.2$

$K_I = 0.008$

$K_D = 0$

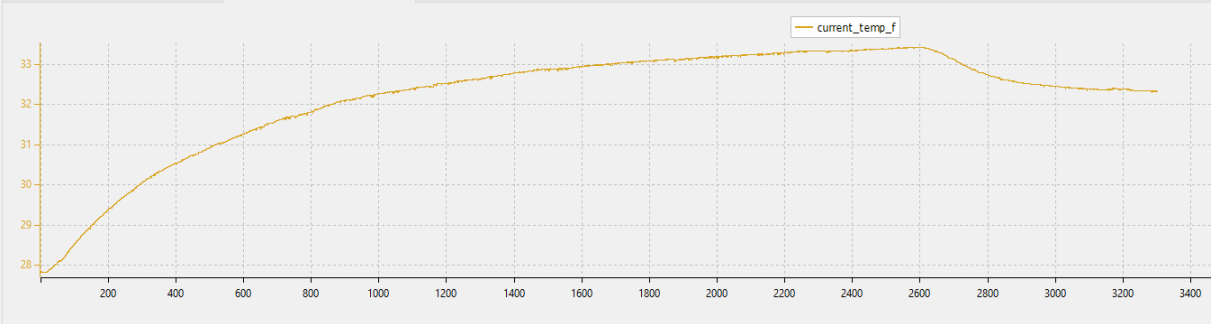
Następnie nastawy zostały zaimplementowane w programie.



Rys. 3 Przebieg sygnału sterującego PWM w Data Trace Timeline Graph

Expression	Type	Value
(x)= pressure	int32_t	100141
(x)= current_temp_f	float	32.5400009
(x)= wartosc_zadana	volatile float	32.3000298
(x)= pwm_duty_u	uint16_t	49041
(x)= pwm_duty_f	float	49041.0703

Na powyższym wykresie widoczny jest przebieg sygnału sterującego PWM na przestrzeni 1450 [s]. Zadana przez nas wartość wynosiła 32.3˚C, zaś najwyższa osiągnięta wartość temperatury to 32.54˚C, zatem uchyb ustalony był na poziomie 0,74% co spełnia założenie projektowe (e_{ust} < 1%). Widoczny na wykresie spadek wartości sygnału PWM w przedziale czasowym <700 ; 750> [s] wynika z osiągnięcia przez rezystor zadanej temperatury, a następnie nagły wzrost wartości wynika z celowego ochłodzenia układu. Dowodzi to prawidłowego działania układu regulacji.

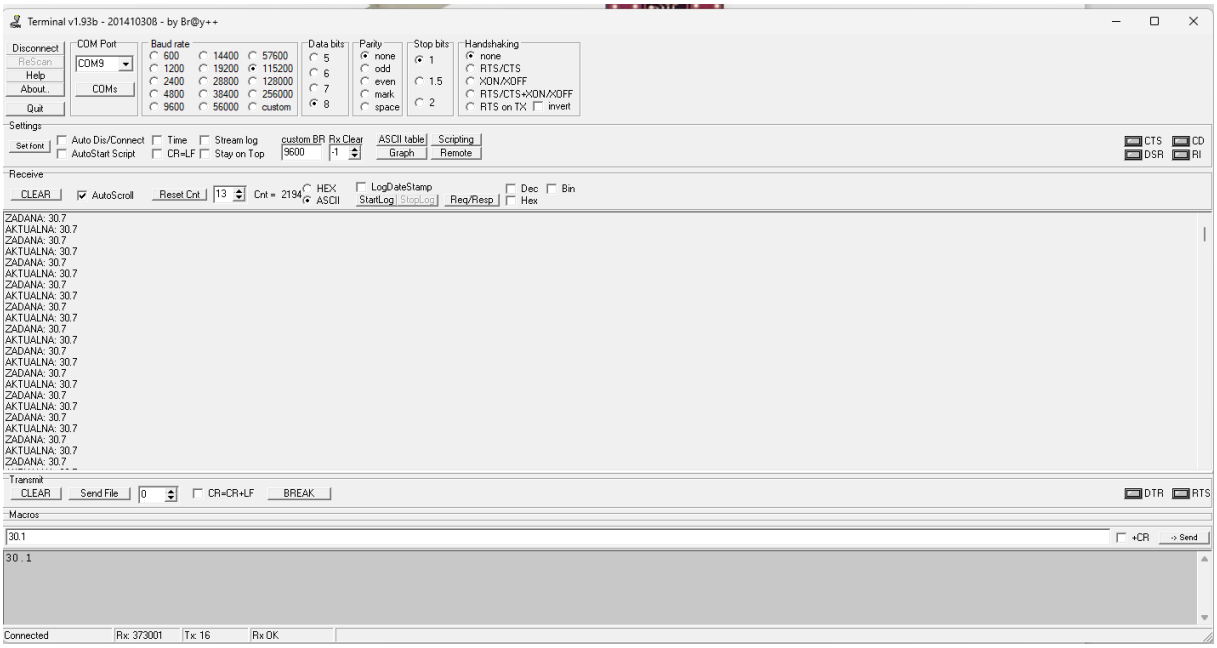


Rys. 4 Przebieg temperatury układu rzeczywistego

Expression	Type	Value
(x)= pressure	int32_t	99693
(x)= current_temp_f	float	32.2999992
(x)= wartosc_zadana	volatile float	32.3000298

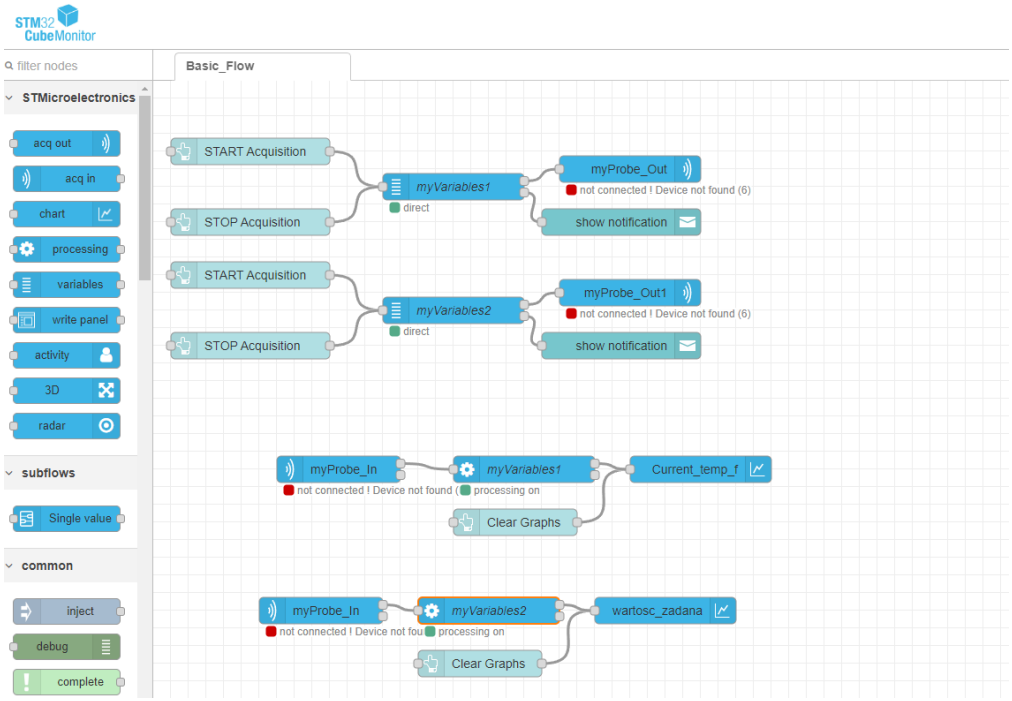
Przy dłuższym czasie nagrzewnia możemy zauważyć, że uchyb maleje jeszcze bardziej. Aktualnie jest na poziomie 0,003% co oczywiście nadal spełnia założenie projektowe.

6. Komunikacja szeregowa (dwukierunkowa)

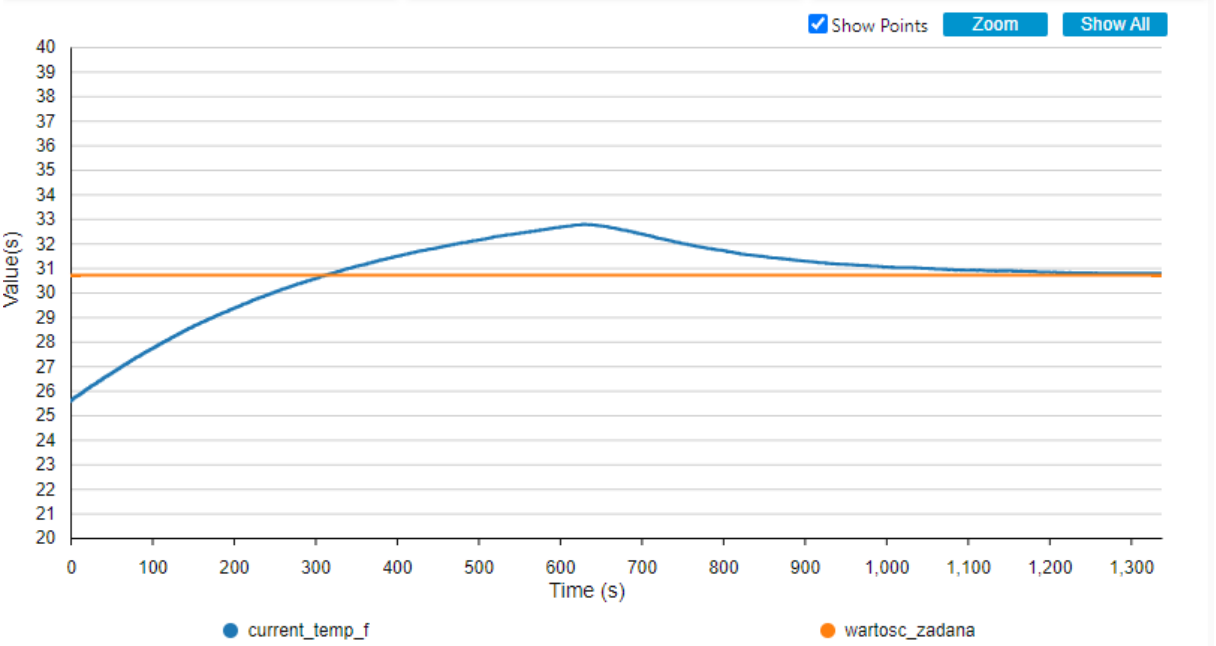


Powyżej znajduje się zrzut ekranu ze środowiska Terminala, gdzie mogliśmy odczytywać wartości temperatury zadanej oraz aktualnej temperatury, która mierzona była poprzez czujnik BMP. Na nagraniu widać, że pokrywa się idealnie z temperaturą wyświetloną na LCD, a także w środowisku STM32CubeIDE – Live Expressions. W konsoli jest możliwość zadania wartości temperatury, do której ma się ustalić.

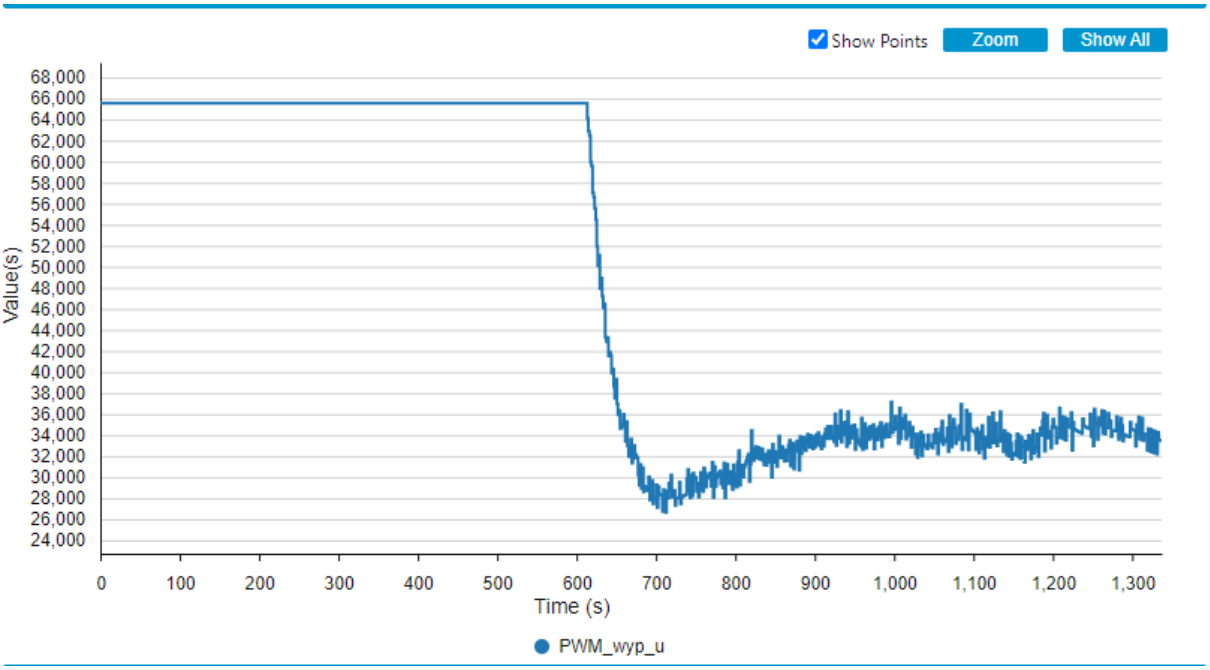
7. Graficzna wizualizacja za pomocą środowiska STM32CubeMonitor.



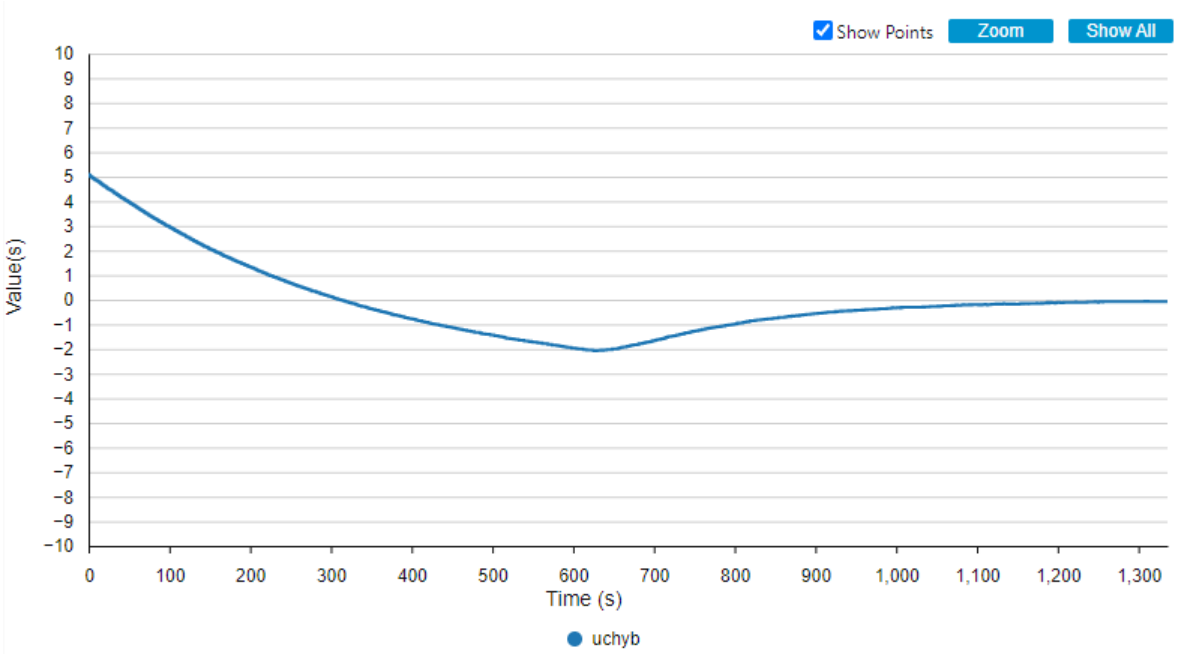
Rys. 5 Schemat blokowy w STM32CubeMonitor



Rys. 6 Wykres przebiegów sygnałów dla temperatury zadanej i aktualnej.



Rys. 7 Wykres przebiegów sygnału dla wartości PWM



Rys. 8 Wykres przebiegu sygnału dla wartości uchybu.

8. Opis rozwiązań zastosowanych w programie mikroprocesora

- **Skonfigurowanie interfejsu I2C** do obsługi czujnika oraz wyświetlacza LED (za pośrednictwem konwertera I2C)

- **Konfiguracja timerów:**

  - ❖ TIM1 został skonfigurowany do obsługi sygnału PWM sterującego bramką użytego tranzystora NPN;
  - ❖ TIM3 został użyty w trybie przerwaniowym NVIC do wysyłania danych i obliczania sygnału sterującego.

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_Base_Start_IT(&htim3);

HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
HAL_TIM_OC_Start(&htim1, TIM_CHANNEL_1);

BMP280_Init(&hi2c4, BMP280_TEMPERATURE_16BIT, BMP280_STANDARD, BMP280_FORCEDMODE);
```

- **Konfiguracja wyświetlacza LCD z wykorzystaniem konwertera I2C**

```
disp.addr = (0x27 << 1);
disp.bl = true;
lcd_init(&disp);
```

```
while (1)
{

// Inicjalizacja wartości początkowej z pomocą przycisku USER i RESET
    if(HAL_GPIO_ReadPin(USER_Btn_GPIO_Port, USER_Btn_Pin) == GPIO_PIN_SET)
    {
        wartosc_zadana+=0.1f;
        dtostrf(wartosc_zadana, 3, 1, (char *)msg);
        sprintf((char *)disp.f_line,"ZADANA:   %s", (char *)msg); //LCD
        HAL_Delay(50);
    }

      dtostrf(current_temp_f, 3, 1, (char *)msg2);

      HAL_UART_Transmit(&huart3, (uint8_t*)msg, strlen(msg), 1000);
      HAL_UART_Transmit(&huart3, (uint8_t*)msg2, strlen(msg2), 1000);

      sprintf((char *)disp.s_line,"AKTUALNA: %s", (char *)msg2); //LCD

      lcd_display(&disp);
      HAL_Delay(50);
```

- **Implementacja regulatora PID**

```
struct Controller{
    float Kp;
    float Ki;
    float Kd;
    float Tp;
    float prev_error;
    float prev_u_I;

};
```

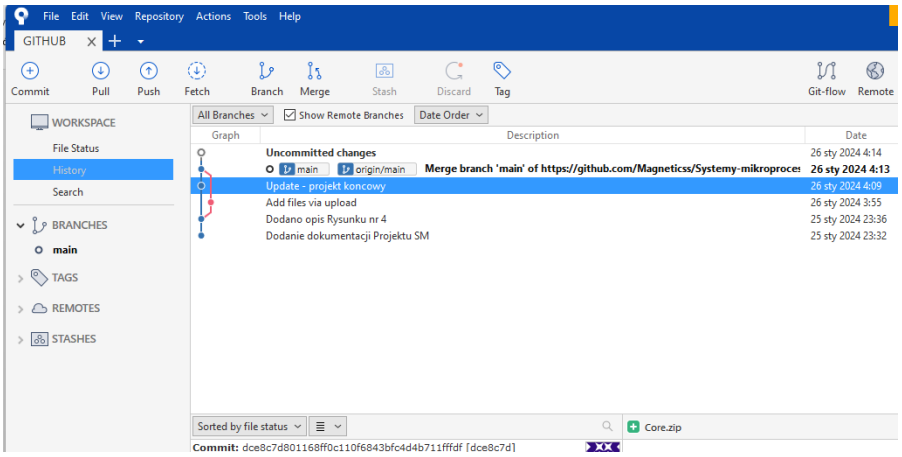
```
 PID1.Kp = 1.2;
 PID1.Ki = 0.008;
 PID1.Kd = 0;
 PID1.Tp = 1;
 PID1.prev_error = 0;
 PID1.prev_u_I = 0;
```

```c
float calculate_PID(struct Controller *PID, float set_temp, float meas_temp){
    float u = 0;
    float error;
    float u_P, u_I , u_D;

    error = set_temp - meas_temp;

    // Proportional
    u_P = PID->Kp * error;

    // Integral
    u_I = PID->Ki * PID->Tp / 2.0 * (error + PID->prev_error) + PID->prev_u_I;
    PID->prev_u_I = u_I;

    // Derivative
    u_D = (error - PID->prev_error) / PID->Tp;

    PID->prev_error = error;

    // Sum of P, I and D components
    u = u_P + u_I + u_D;

    return u;
}
```

```c
if(PWM_wyp_float < 0.0) PWM_wyp_u = 0;
else if(PWM_wyp_float > htim1.Init.Period) PWM_wyp_u = htim1.Init.Period;
else PWM_wyp_u = (uint16_t) PWM_wyp_float;

PWM_wyp_u = htim1.Init.Period;      // 100% PWM duty for creating model
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, PWM_wyp_u);
```

9. GITHUB – Kontrola wersji

Jednym z dodatkowych aspektów, jakie wykorzystaliśmy w naszym projekcie jest kontrola wersji w środowisku GITHUB. Poniżej został podany link, dzięki któremu mamy podgląd końcowy, co działo się w projekcie ku końcu. Skorzystaliśmy w tym celu z narzędzia/ środowiska Sourcetree, którego screen jest poniżej.



10. Komentarz do video.

Na początku za pomocą przycisku na płytce Nucleo zwiększaliśmy temperaturę zadaną. W pierwszym momencie widać jak żółta dioda miga – świadczy to o tym, że wartość PWM jest równa 0. Po stopniowym wzroście temperatury, kolejno zapala się dioda żółta, niebieska, czerwona i zielona. Przez chwilę widać jak gaśnie czerwona, a pozostałe dwie świecą – oznacza to, że wartość PWM zmniejszyła się. W międzyczasie pokazujemy zmienne w środowisku STM32CubeIDE, w zakładce Live Expression – są zgodne z wartościami wyświetlanymi na LCD – tj. zadana(„wartosc_zadana") oraz aktualna(„current_temp_f"). Następnie używamy przycisku Reset do wyzerowania wartości, co widać zarówno na LCD, jak i w programie. W kolejnym kroku przechodzimy do Terminala, gdzie mamy możliwość podglądu temperatury zadanej oraz aktualnego jej pomiaru z czujnika. Następnie w konsoli wpisujemy wartość zadaną temperatury, która wyświetla się odpowiednio na LCD, jak i w Live Expressions. Przechodzimy do środowiska STM32CubeMonitor. Za jego pomocą obserwujemy 3 różne wykresy na żywo: Wykres temperatury zadanej oraz aktualnej, PWM oraz uchyb.

11. Bibliografia.

- Materiały udostępnione poprzez eKursy z Laboratorium SM
- Materiały udostępnione poprzez eKursy z Wykładów SM
- Kanał Piotra Duby na platformie Youtube
- Nauka obsługi kontroli wersji w środowisku Github:

12. Linki do nagrania oraz GitHub'a

- **https://github.com/Magneticss/Systemy-mikroprocesorowe.git**
- **https://drive.google.com/drive/folders/17HiZqeP5C_vBosM-p8brPbFkpm8MTS4H?usp=sharing**