Systemy mikroprocesorowe

Sprawozdanie – Regulator Temperatury PID

Regulator Temperatury PID

Naskręt,Naumienko

1 Cel projektu

Celem projektu było zaprojektowanie i zrealizowanie mikroprocesorowego układu regulacji temperatury w zamkniętej pętli sprzężenia zwrotnego. Projekt zakładał wykorzystanie mikrokontrolera z rodziny STM32, cyfrowego czujnika temperatury oraz algorytmu PID do sterowania elementem grzejnym. Dodatkowymi wymaganiami była realizacja interfejsu użytkownika (wyświetlacz LCD, enkoder) oraz dwukierunkowej komunikacji szeregowej (UART) do celów telemetrycznych.

2 Wykorzystane elementy i konfiguracja sprzętowa

2.1 Elementy składowe

W projekcie wykorzystano następujące podzespoły:

- **Mikrokontroler:** STM32 Nucleo.
- **Czujnik pomiarowy:** BMP280 (Interfejs I2C) – pomiar temperatury otoczenia.
- **Element wykonawczy:** Rezystor mocy sterowany tranzystorem (Grzałka).
- **Interfejs HMI:** Wyświetlacz LCD 2x16 (sterownik I2C HD44780) oraz enkoder inkrementalny.

2.2 Konfiguracja Pinout

Tabela 1 przedstawia przypisanie wyprowadzeń mikrokontrolera do poszczególnych peryferiów.

Tabela 1: Konfiguracja wyprowadzeń mikrokontrolera

Peryferium	**Pin / Port**	**Funkcja**
Grzałka	PD7	Wyjście sterujące (On-Off / PWM programowy)
BMP280	I2C1 (PB8, PB9)	Komunikacja z czujnikiem temperatury
LCD 2x16	I2C4	Obsługa wyświetlacza
Enkoder	TIM1	Ustawianie temperatury zadanej
UART	USART3	Komunikacja z PC (115200 baud)
LED Status	PB0 (LD1)	Sygnalizacja grzania

3 Wykorzystane środowiska programistyczne

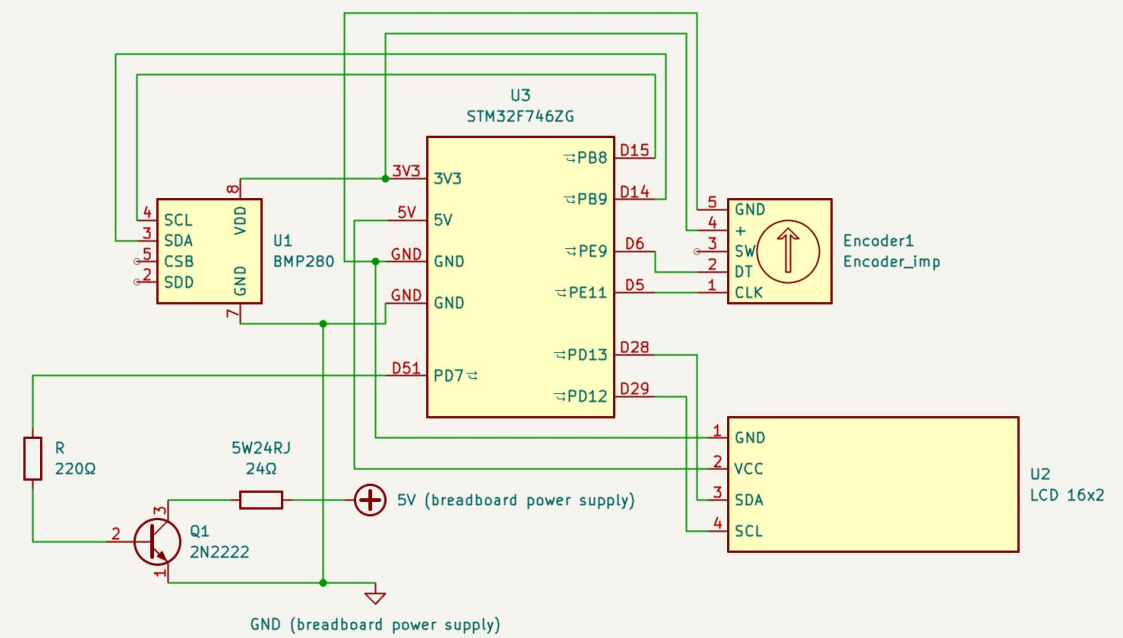
3.1 STM32 CubeIDE

STM32 CubeIDE to darmowe środowisko programistyczne firmy STMicroelectronics przeznaczone do tworzenia aplikacji na mikrokontrolery STM32. Łączy w jednym narzędziu edytor kodu, kompilator, debugger oraz moduł STM32CubeMX, który umożliwia graficzną konfigurację pinów, zegarów i peryferiów mikrokontrolera. Środowisko bazuje na platformie pozwala na programowanie w języku C oraz C++, z wykorzystaniem bibliotek HAL i LL. Dzięki wbudowanej obsłudze programatora i debuggera ST-LINK ułatwia uruchamianie, testowanie i analizę działania aplikacji na sprzęcie.

3.2 Visual Studio Code

Visual Studio Code to darmowe środowisko programistyczne stworzone przez firmę Microsoft. Jest to edytor kodu z rozbudowanymi funkcjami i wieloma rozszerzeniami. Obsługuje wiele języków programowania, w tym użyty do stworzenia charakterystyki regulacji 2 Python. VS Code oferuje m.in. podświetlanie składni, inteligentne podpowiedzi, debugowanie oraz integrację z systemami kontroli wersji, takimi jak Git. Głównymi zaletami tego środowiska jest jego prostota i elastyczność.

4 Schemat elektroniczny układu pomiarowego



Rysunek 1: Schemat badanego układu

5 Algorytm sterowania (PID)

W projekcie zaimplementowano dyskretny regulator PID. Wyjście regulatora steruje czasem włączenia grzałki w cyklu 1-sekundowym (Time Proportional Control).

Równanie regulatora w dziedzinie czasu ciągłego:

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

gdzie: $e(t)$ to uchyb regulacji (różnica między wartością zadaną a mierzoną).

5.1 Implementacja programowa

Poniżej przedstawiono kluczowy fragment kodu odpowiedzialny za obliczanie nastaw regulatora. Zastosowano zabezpieczenie przed nasyceniem członu całkującego (Anti-Windup Clamp).

Listing 1: Struktura i funkcja obliczająca PID

```
typedef struct {
    float Kp;
    float Ki;
    float Kd;
    float prevError;
    float integral;
    float outMin;
    float outMax;
```

```
} PID_Controller;

PID_Controller pid = {
    .Kp = 400.0f,
    .Ki = 10.0f,
    .Kd = 50.0f,
    .prevError = 0.0f,
    .integral = 0.0f,
    .outMin = 0.0f,
    .outMax = 1000.0f
};

float PID_Compute(PID_Controller *pid, float setpoint, float measured)
{
    float error = setpoint - measured;

    // Człon Proporcjonalny
    float P = pid->Kp * error;

    // Człon Całkujący
    pid->integral += error;
    if (pid->integral > pid->outMax) pid->integral = pid->outMax;
    else if (pid->integral < pid->outMin)
        pid->integral = pid->outMin;

    float I = pid->Ki * pid->integral;

    // Człon Różniczkujący
    float D = pid->Kd * (error - pid->prevError);
    pid->prevError = error;

    // Suma
    float output = P + I + D;

    // Ograniczenie wyjścia (Clamp)
    if (output > pid->outMax) output = pid->outMax;
    else if (output < pid->outMin) output = pid->outMin;

    return output;
}
```

6 Analiza modelu i dobór nastaw

Do wyznaczenia nastaw regulatora wykorzystano zmodyfikowaną **II Metodę Zieglera-Nicholsa** (metodę oscylacji krytycznych). Ze względu na bezpieczeństwo układu, parametry estymowano na podstawie oscylacji gasnących w pobliżu punktu pracy, a nie doprowadzając

układ do trwałej niestabilności.

6.1 Identyfikacja parametrów dynamicznych

Na podstawie zarejestrowanej charakterystyki skokowej (Rys. 2) wyznaczono naturalny okres oscylacji układu.

- Czas wystąpienia pierwszego szczytu: $t_1 \approx 400\,s$
- Czas wystąpienia drugiego szczytu: $t_2 \approx 1000\,s$
- Okres oscylacji (T_{osc}): $1000\,s - 400\,s = 600\,s$
- Wzmocnienie, przy którym wystąpiły oscylacje (K_p): ≈ 400

6.2 Obliczenia analityczne

Przyjmując $T_{osc} = 600\,s$ jako bazę do obliczeń (zgodnie z regułami Z-N dla regulatora PID):

- **Czas zdwojenia (T_i):** $0.5 \cdot T_{osc} = 300\,s$.
- **Czas wyprzedzenia (T_d):** $0.125 \cdot T_{osc} = 75\,s$.

Przeliczenie na wzmocnienia dla algorytmu dyskretnego (przyjmując czas próbkowania $T_s = 1\,s$):

$$K_i(obl) = \frac{K_p \cdot T_s}{T_i} = \frac{400 \cdot 1}{300} \approx 1.33 \tag{2}$$

$$K_d(obl) = \frac{K_p \cdot T_d}{T_s} = \frac{400 \cdot 75}{1} = 30000 \tag{3}$$

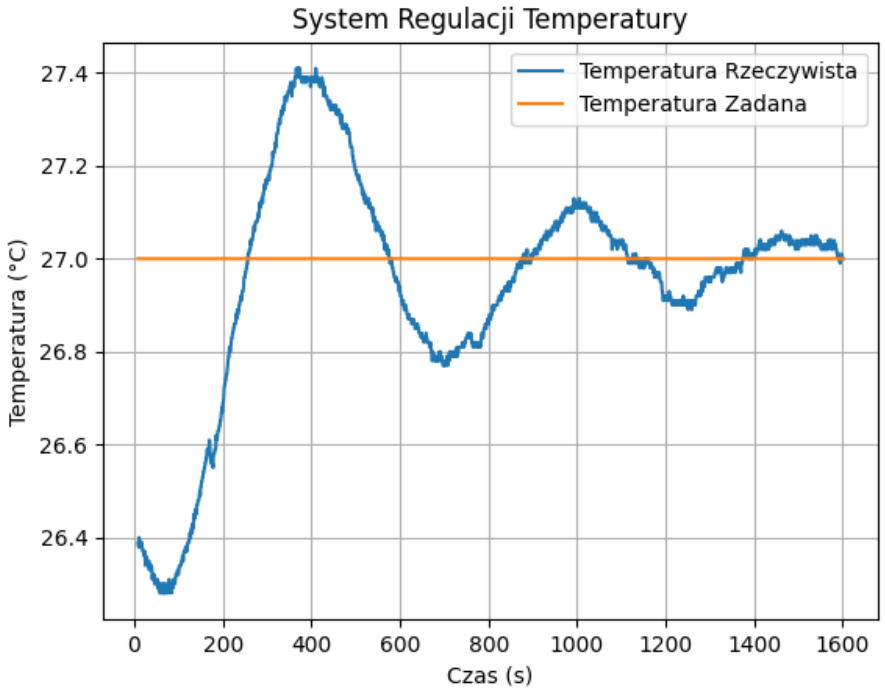
6.3 Korekta eksperymentalna (Fine-tuning)

Obliczone wartości teoretyczne wymagały korekty ze względu na specyfikę obiektu (duża bezwładność cieplna) oraz charakter cyfrowy (szumy kwantyzacji):

1. **Zwiększenie K_i (do 10.0):** Teoretyczna wartość (1.33) powodowała zbyt wolną redukcję uchybu statycznego.
2. **Zmniejszenie K_d (do 50.0):** Teoretyczna wartość (30000) była zbyt wysoka dla implementacji dyskretnej, powodując silne wzmacnianie szumów pomiarowych czujnika BMP280, co prowadziło do niestabilnego sterowania grzałką.

7 Wyniki i weryfikacja

Poniższy wykres przedstawia charakterystykę regulacji dla zadanej temperatury $27.0°C$.



Rysunek 2: Przebieg regulacji temperatury. Linia pomarańczowa - wartość zadana, linia niebieska - temperatura mierzona.

Wykres wskazuje na stabilną pracę układu. Występują oscylacje gasnące (przeregulowanie), co jest charakterystyczne dla przyjętych nastaw, jednak układ skutecznie sprowadza temperaturę do wartości zadanej. Po ustabilizowaniu się przebiegu, błąd regulacji (uchyb) utrzymywał się na niskim poziomie wynoszącym ok. 0.3%–0.5%.

8 Instrukcja obsługi i funkcjonalności

1. Po uruchomieniu układ wyświetla na LCD aktualną temperaturę oraz zadaną wartość (T_{zad}).
2. **Zmiana lokalna:** Obrót enkoderem powoduje zmianę temperatury zadanej.
3. **Sterowanie zdalne (UART):** Podłączenie terminala (115200 baud) umożliwia:
   - Odbiór danych telemetrycznych (format: `temp_akt temp_zad`).
   - Zmianę nastaw komendami: `+` (zwiększ), `-` (zmniejsz).
4. Dioda LED na nucleo sygnalizuje momenty załączenia elementu grzejnego.

9 Podsumowanie

Układ poprawnie odczytuje temperaturę, a zaimplementowany algorytm PID skutecznie steruje procesem grzania. Zastosowanie metody Zieglera-Nicholsa pozwoliło na wstępne oszacowanie nastaw, a późniejsza korekta eksperymentalna wyeliminowała problemy związane z szumami cyfrowymi oraz dużą inercją termiczną obiektu.