POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI
INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ
ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PROJEKT

Systemy mikroprocesorowe (laboratorium) [WARiE_2021-22_AiR_Dz_1_5_D_LUCZAK_21/22]

(Pełna nazwa kursu w systemie eKursy)

Regulator temperatury

(TEMAT PROJEKTU)

Karol Dębski, Paweł Michalak, Mateusz Skierski

(Autorzy)

Prowadzący:
dr inż. Dominik Łuczak
dominik.luczak@put.poznan.pl

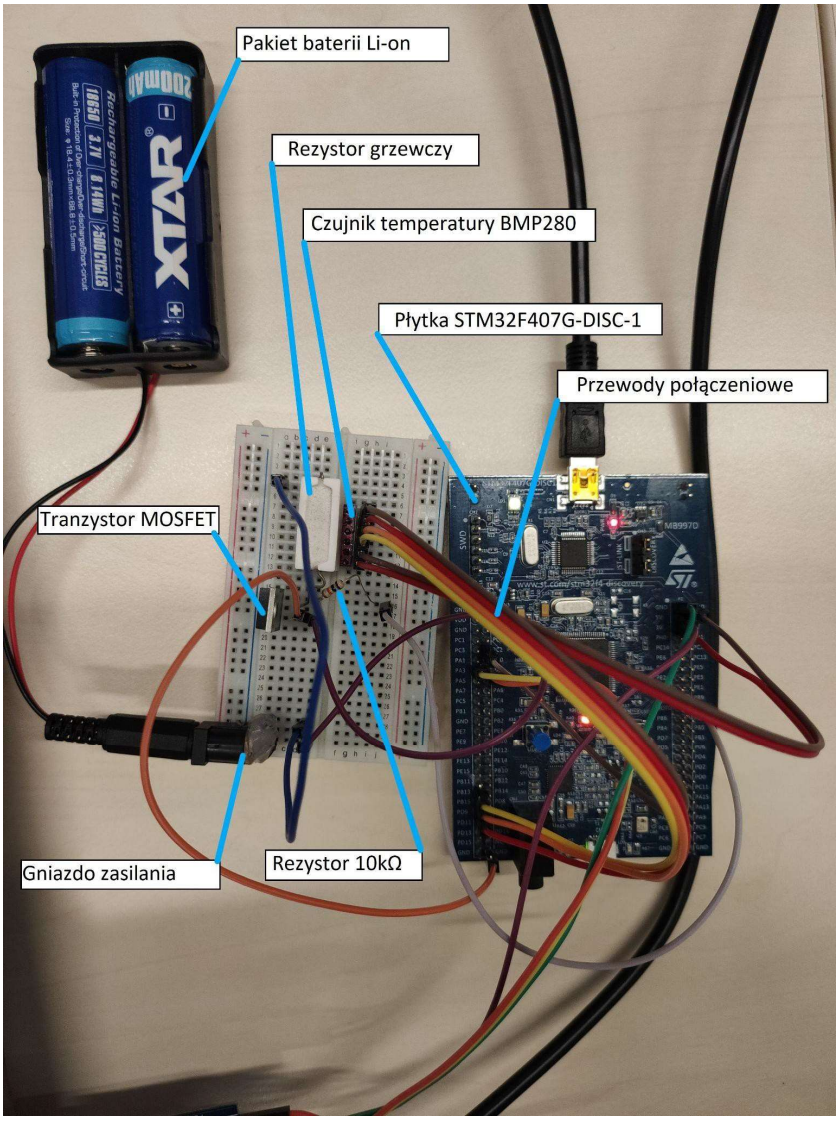
1. Specyfikacja

Układem regulacji jest rezystor grzejny sterowany za pomocą tranzystora MOSFET. Czujnik temperatury znajduje się pod rezystorem. Do zadawania temperatury używany jest USART. Pomiar aktualnej temperatury jest wykonywany co 1 sekundę. Co jedną sekundę wysyłana jest wartość aktualnej temperatury, wartość zadanej temperatury oraz wartość sygnału zadanego. W układzie regulacji został użyty regulator PID z anti-windupem w postaci odcinania. Układ reguluję temperaturę rezystora w zakresie od 23.5 do 45 stopni.

Użyte elementy:

- płytka rozwojowa STM32F407G-DISC-1
- rezystor grzejny 20 Ω, 5 W
- przewody połączeniowe
- zestaw 2 ogniw litowo-jonowych połączonych szeregowo
- tranzystor MOSFET IRL540N
- rezystor 10 kΩ
- czujnik temperatury i ciśnienia BMP280

-

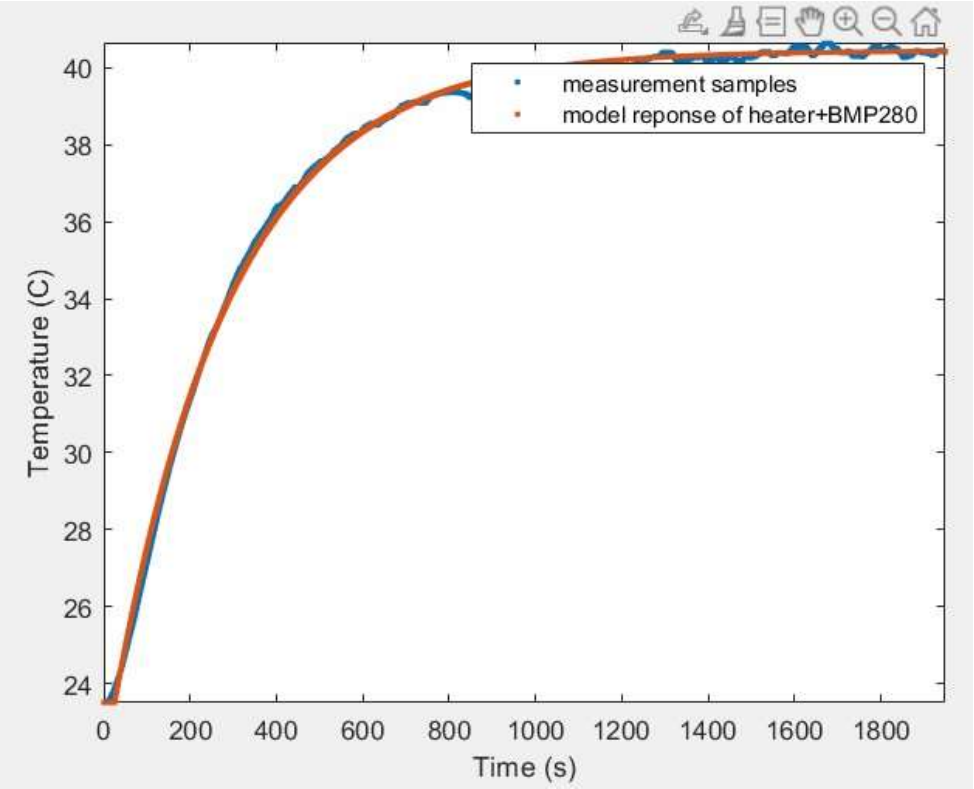


Rys 1. Zdjęcie zbudowanego układu wraz z opisem

2. Implementacja

Listing 1 Kod odpowiadający za identyfikację obiektu (Matlab)

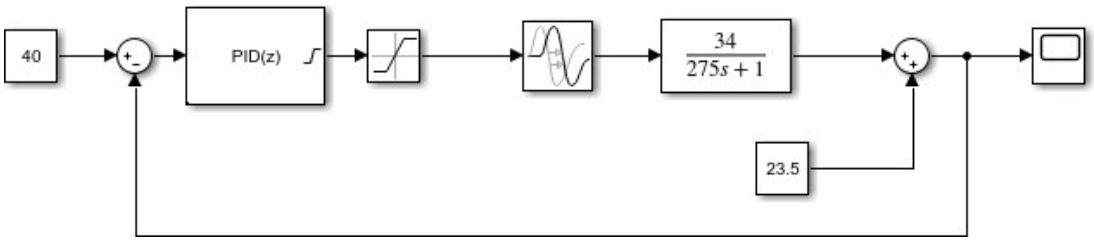
```
temperature = load('data_pid_controller_20211220-035636.txt');
dt=1; %one second
number_of_samples = length(temperature);
t = (0:number_of_samples-1)*dt;
%Input signal
input_amplitude = 0.5; % 0.9 -> 90% of PWM duty (PWM1=90;)
input = input_amplitude*ones( 1 ,number_of_samples);
%LTI model (linear time-invariant model)
s = tf('s');
k = 16.95/input_amplitude;  %model gain
T = 1948-1590-83;%72 323  %model time constant
delay=25;  %model delay
H = k/(1+s*T)*exp(-s*delay); % model
disp(sprintf('Model parameters k=%.2g, T=%g, delay=%g\n', k, T, delay));
%Model response
model_response = lsim(H,input,t);
model_response = model_response + 23.5; %add offset
%Model error
residuum = temperature - model_response';
error_abs_sum = sum(abs(residuum));
disp(sprintf('Model error sum(abs(residuum)) = %g\n', error_abs_sum));
figure(1);
plot(t,temperature, '.', t, model_response, '.');
title('(author: D. Luczak)');
xlabel('Time (s)');
ylabel('Temperature (C)');
%k=34, T=275, delay=25
legend('measurement samples', 'model reponse of heater+BMP280');
axis tight;
figure(2);
plot(t,residuum, '.');
title('Residuum (measurement samples - model reponse)');
xlabel('Time (s)');
ylabel('Temperature (C)');
axis tight;
ylim([-0.5 0.5]);
[Gsnum, Gsden]=tfdata(H);
%Conversion to discrete model
H_discrete = c2d(H, dt, 'tustin');
disp(H_discrete)
%Second order sections
[sos, g] = tf2sos(cell2mat(H_discrete.num), cell2mat(H_discrete.den));
disp('gain:');
disp(g);
disp('sos matrix:');
disp(sos);
```



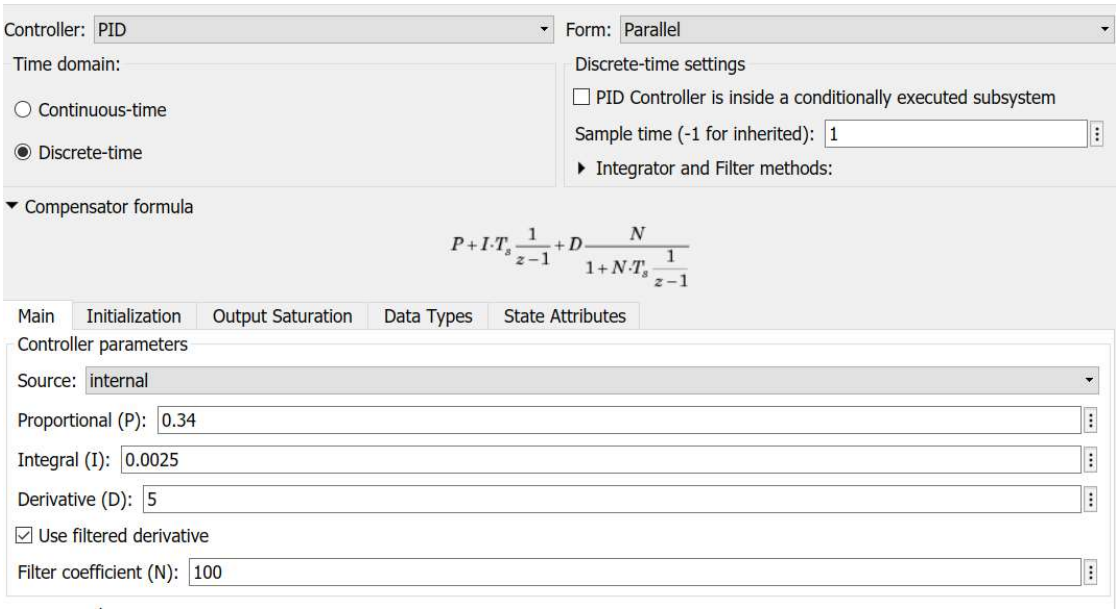
Rys 2. Odpowiedź obiektu (niebieski - przebieg dla rzeczywistego układu, czerwony - przebieg dopasowany)

W pierwszym kroku zbierane są dane dla wypełnienia 50% PWM sygnału sterującego tranzystorem. Następnie dobierane są parametry modelu w środowisku Matlab, tak by przebieg wpasował się w dane (Listing 1, Rys 2).

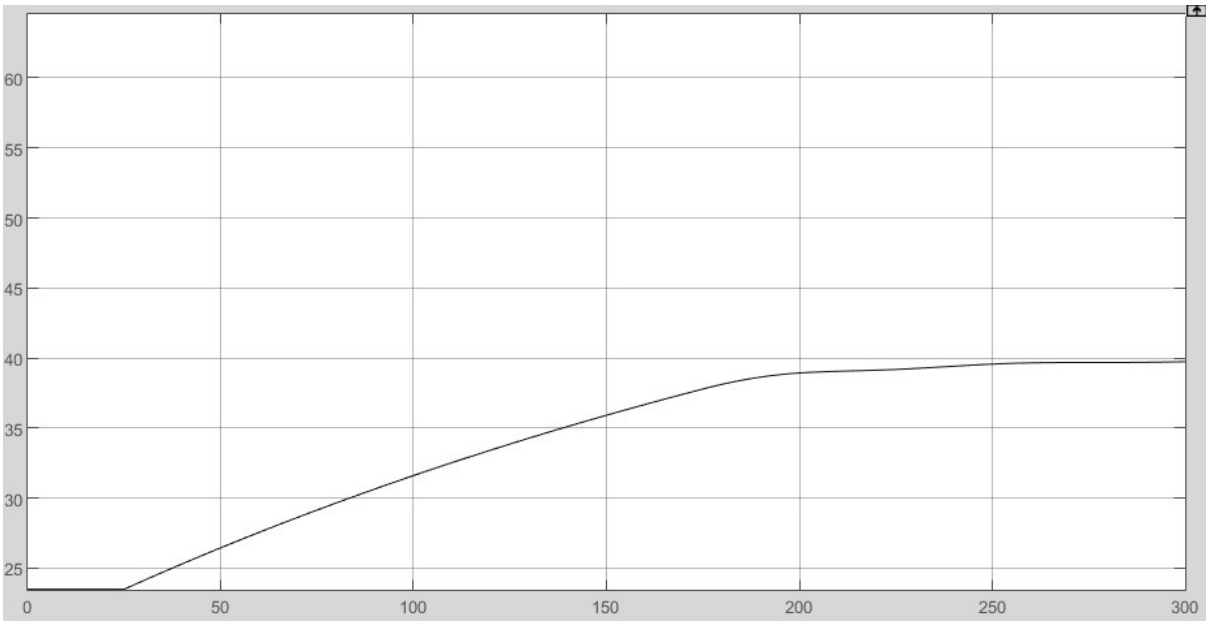
Na podstawie odpowiedzi obiektu można stwierdzić że mamy do czynienia z obiektem inercyjnym z opóźnieniem (Rys 2).



Rys.3 Układ zaprojektowany w środowisku Matlab



Rys.4 Nastawy regulatora PID uzyskane za pomocą metody prób i błędów



Rys 5. Odpowiedź układu po przeprowadzeniu symulacji w programie Simulink

Do symulacji działania obiektu uzyskanego ze skryptu wykorzystano oprogramowanie Simulink. Zbudowany układ znajduje się na Rys 3. Nastawy PID zostały dobrane w sposób eksperymentalny i uzyskano następujące wyniki które można zobaczyć na Rys 4. Spodziewana odpowiedź układu rzeczywistego jest przedstawiona na Rys 5.

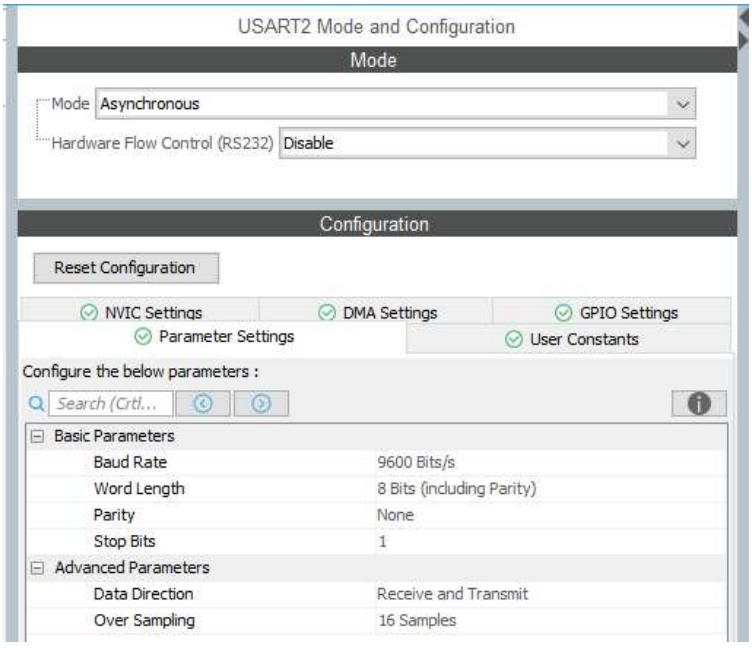
Listing 2 Zmienne globalne oraz struktury

```
uint8_t bufforRx[BUFFOR_RX_SIZE];
float feedback_mesaurement=0;
float previous_pid_output;
float u=0, P, I, D, error, integral, derivative;
float t=0;
float dt=1;
uint16_t start_measurement=0;
typedef struct{float Kp;float Ki;float Kd;float Kc;float dt;}pid_parameters_t;
typedef struct{pid_parameters_t p;float previous_error,previous_integral,previous_pid_output;}PID_t;
float set_point=23.5;
float temperature;
float pid_output=0;
float pwm_duty;
```

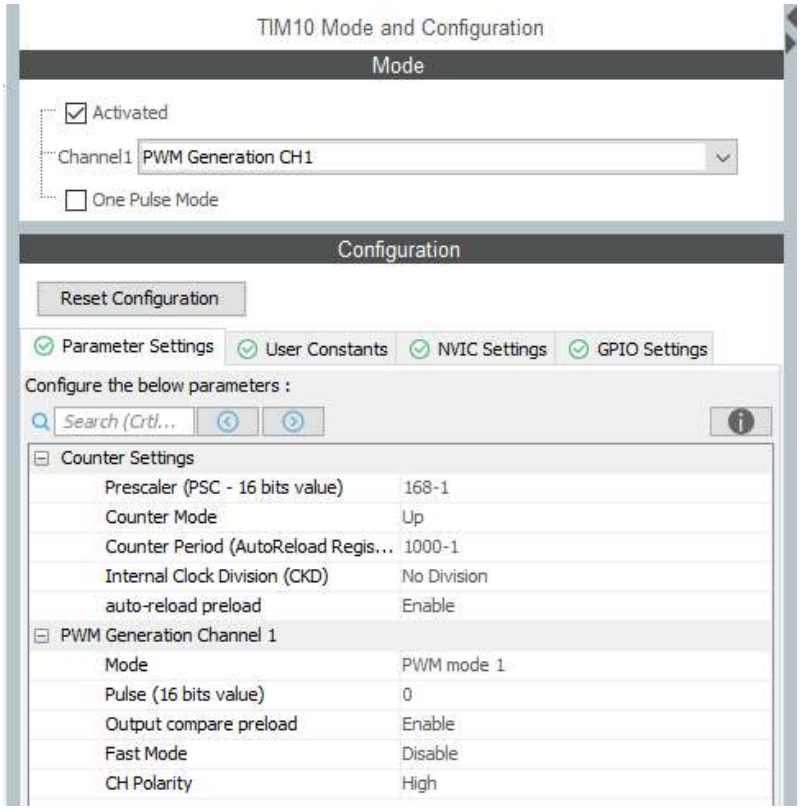
Listing 3 Funkcja obliczająca sygnał wyjściowy PID

```
/* USER CODE BEGIN PFP */
float calculate_discrete_pid(PID_t* pid, float setpoint, float measured){
        error = setpoint-measured;
        P = pid->p.Kp * error;
        if(pid->previous_pid_output<1 && pid->previous_pid_output>0)
        {
                integral = pid->previous_integral + (error+pid->previous_error);
                pid->previous_integral = integral;
                I = (pid->p.Ki)*integral*(pid->p.dt/2.0);
        }
        derivative=(error-pid->previous_error)/pid->p.dt; //numerical derivative without filter
        pid->previous_error = error;
        D = pid->p.Kd*derivative;
        u = P + I + D;
        pid->previous_pid_output=u;
        return u;
}
PID_t pid1={.p.Kp=0.34,.p.Ki=0.0025,.p.Kd=5,.p.Kc=0.001,.p.dt=1};
```

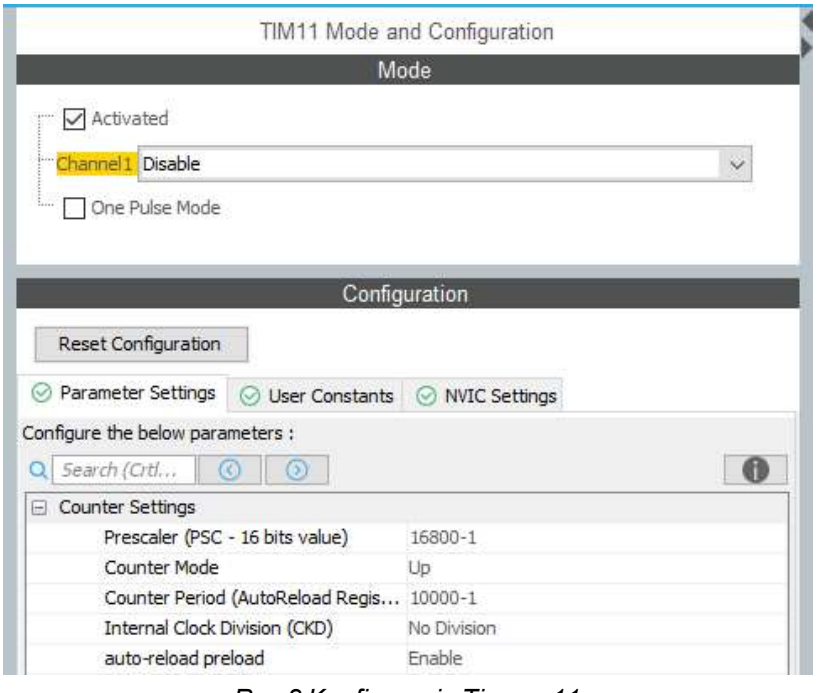
Listing 3 tworzy zmienne globalne oraz struktury, natomiast w *Listingu 4* jest umieszczona funkcja implementująca dyskretny regulator PID, który pobiera sygnał zadany oraz zwraca wartość sygnału na wyjściu.



Rys 6. Konfiguracja UART



Rys. 7 Konfiguracja Timera 10 (jako PWM)



Rys.8 Konfiguracja Timera 11

Listing 4 Funkcja przerwania UART

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){
	if(huart->Instance == USART2){
		float tmp=0;
		sscanf((char *)bufforRx,"%f",&tmp);
		if(tmp>23.5 && tmp<45.0){
			set_point=tmp;
		}else{
			set_point=23.5;
		}
	}
	memset(bufforRx, 0, BUFFOR_RX_SIZE);
	HAL_UARTEx_ReceiveToIdle_IT(&huart2, bufforRx, BUFFOR_RX_SIZE);
}
```

Listing 5 Funkcja przerwania dla TIM11

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
	if(htim->Instance == TIM11){
		feedback_mesaurement=(float)temperature;
		HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);
		pid_output=calculate_discrete_pid(&pid1,set_point,feedback_mesaurement);
		previous_pid_output=pid_output;
		if(pid_output>1.0) pid_output=1.0;
		if(pid_output<0) pid_output=0;
		pwm_duty=(uint16_t)(999.0*pid_output);
		__HAL_TIM_SET_COMPARE(&htim10,TIM_CHANNEL_1,pwm_duty);
		start_measurement=1;
	}
}
```

Listing 6 Funkcja służąca do wysyłania danych, które następnie są odczytywane z poziomu terminalu

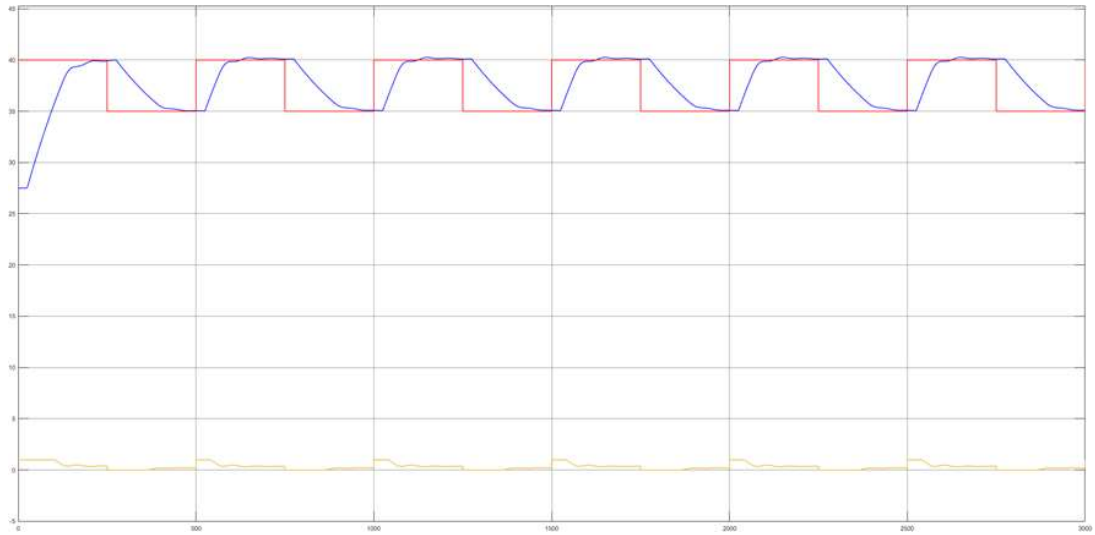
```
void BMP2_user_app_print_sensor_data(struct bmp2_data *comp_data)
{
	uint8_t uart_data[200];
	uint16_t uart_size;
	uart_size = sprintf((char*)uart_data, "Temperature: %0.2f || Set point: %0.2f || Pid output:%0.2f\r\n"
,comp_data->temperature,set_point,pid_output);

	HAL_UART_Transmit(&huart2, uart_data, uart_size, 0xffff);

}
```

Listing 7 Pętla while w funkcji głównej Main

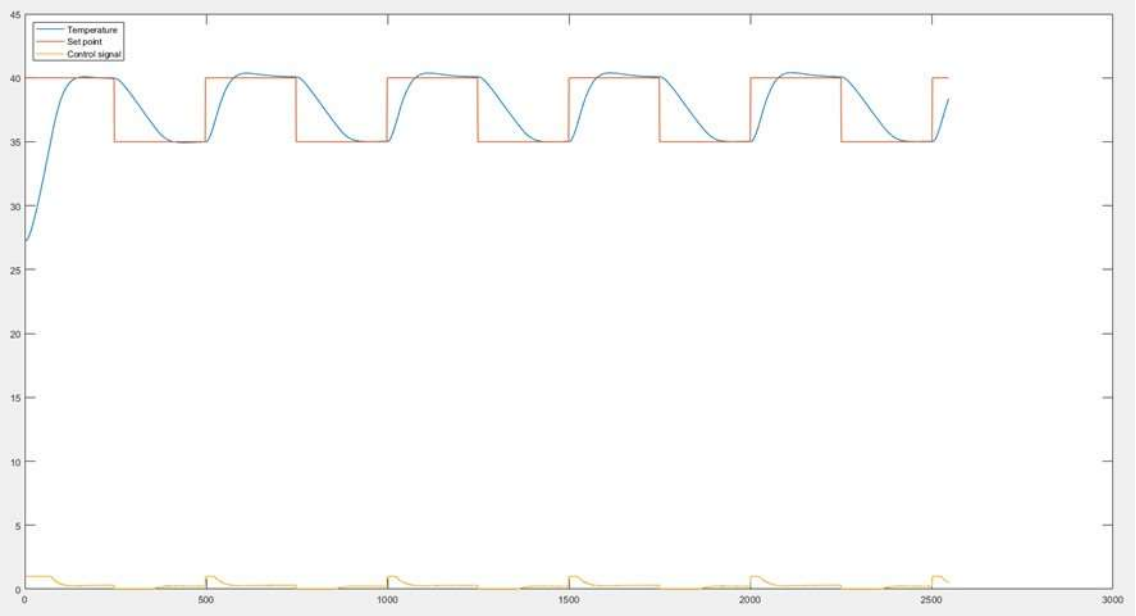
```
BMP2_user_app_configuration(&dev);
 HAL_UARTEx_ReceiveToIdle_IT(&huart2, bufforRx, BUFFOR_RX_SIZE);
 HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
 HAL_TIM_Base_Start_IT(&htim11);
 __HAL_TIM_SET_COMPARE(&htim10,TIM_CHANNEL_1,0);
 BMP2_user_app_stream_sensor_data_by_uart(&dev);

 while (1)
 {
	if(start_measurement){
		BMP2_user_app_stream_sensor_data_by_uart(&dev);
		start_measurement=0;
	}
   /* USER CODE END WHILE */
   MX_USB_HOST_Process();

   /* USER CODE BEGIN 3 */
 }
```

Co jedną sekundę wywoływany jest callback dla TIM11 (Listing 5). W nim obliczany jest sygnał sterujący oraz na podstawie jego ustawiany jest rejestr pośrednio odpowiedzialny za sygnał PWM. Zmieniana jest także zmienna start_measuremet która zezwala na dokonanie pomiaru w pętli głównej programu (Listing 7). Wraz z dokonaniem pomiaru wysyłane są wartości aktualnej temperatury, temperatury zadanej oraz wartości sygnału sterującego (Listing 6). W każdej chwili możliwe jest wysłanie za pomocą UARTA wartości zadanej temperatury (Listing 4). Jeśli wartość nie mieści się w przedziale od 23.5 do 45.0 lub wiadomość nie jest liczbą to wartość temperatury zadanej jest ustawiana na 23.5.

3. Wyniki testów

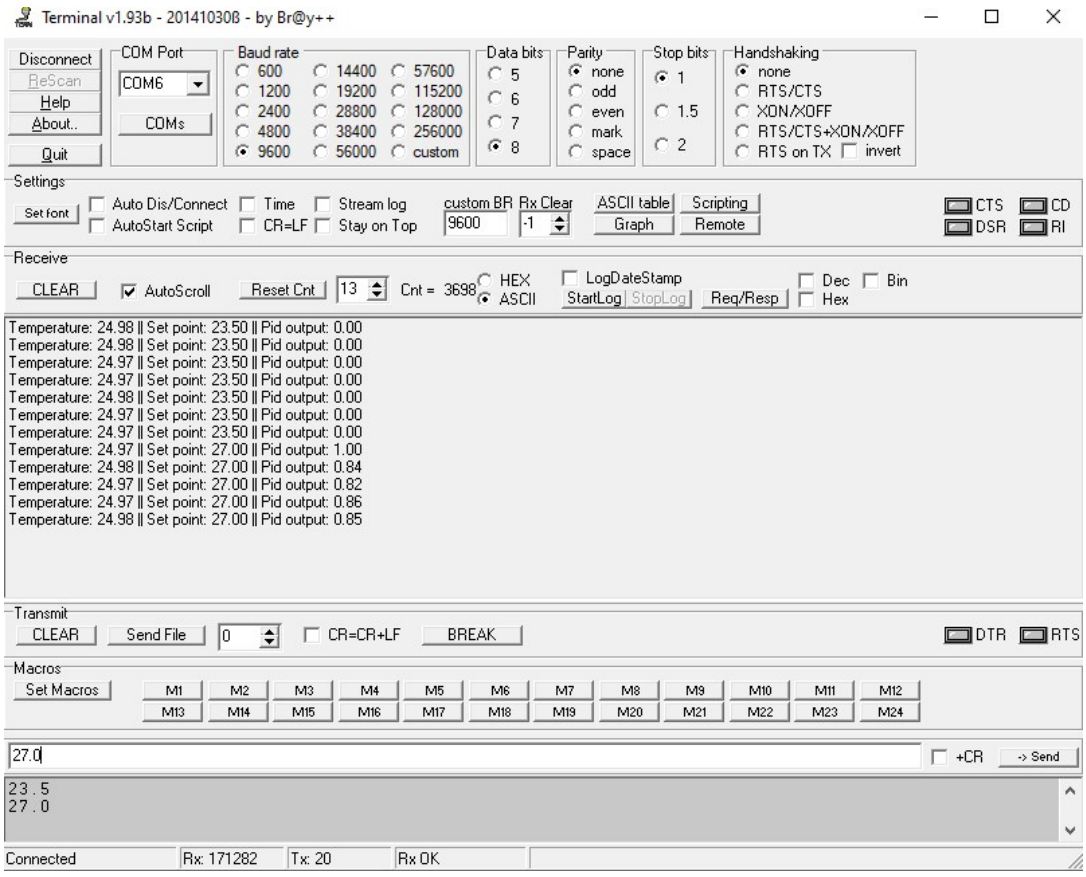


Rys. 9 Wykres wartości zadanej (czerwony), temperatury (niebieski) i sygnału sterującego (żółty) – symulacja w simulinku



Rys. 10 Wykres wartości zadanej, temperatury i sygnału sterującego – obiekt rzeczywisty

Na rysunkach powyżej przedstawiono odpowiedzi układu symulacyjnego (Rys. 6) oraz układu rzeczywistego (Rys. 7). Analizując odpowiedzi można zauważyć że pokrywają się one zapewniając uchyb na poziomie niższym niż 5% (<1,075 ºC).



Rys. 11 Terminal z komunikacją odczytu/wysyłu

Wysłanie wartości zadanej temperatury za pomocą terminalu powoduje zmianę parametru zadanej temperatury której odczyt odbywa się za pomocą terminalu.

4. Wnioski

Zrealizowany układ spełnia następujące wymagania:

- System dokonuje pomiaru regulowanej zmiennej ze stałym okresem próbkowania
    - implementacja przedstawiona w ***Listingu 7, Rys. 8, Listing 5***
- System umożliwia sterowanie w bezpiecznym zakresie zmian regulowanej zmiennej
    - implementacja przedstawiona w ***Listingu 4***
- System zapewnia uchyb ustalony na poziomie 5% zakresu regulacji (np. jeżeli przyjmiemy zakres regulacji temperatury jako 20-40ºC, uchyb ustalony w całym przedziale nie może przekraczać 1 oC).
    - wymagany wynik widoczny na ***Rys.10***
- System umożliwia zadawanie wartości referencyjnej za pomocą komunikacji szeregowej
    - wymagany wynik widoczny na ***Rys.11,*** oraz ***Listingu 4***
- System umożliwia podgląd aktualnej wartości sygnału: pomiarowego, referencyjnego i steruj¡cego za pomocą komunikacji szeregowej lub urządzenia wyjścia
    - wymagany wynik widoczny na ***Rys.11,*** oraz ***Listingu 6***