Miernik temperatury 9

Labolatorium Systemów Elektronicznych, Informatyka - I rok, II semestr

Autorzy: Jakub Achtelik, Kacper Oborzyński

Koszalin, 15 Czerwca 2023

Spis treści

1. Projekt

Temat: MIERNIK TEMPERATURY 9
1. Odczyt temperatury z czunika DS18B20
2. Wyświetlanie jej na ekranie LCD
3. Wykorzystanie endokera do ustawienia progu temperatury maksymalnej i minimalnej
4. Zapewnienie sygnalizacji przekroczenia progu

Platforma: Arduino IDEE C/C++

Projekt na symulatorze: **https://wokwi.com/projects/367315813073881089**

1.1 Wykorzystane komponenty

- Arduino Uno - ATmega328
- Miernik temperatury DS18B20 + 2x rezystor 10k Ohm
- Wyświetlacz LCD 20x4
- Enkoder inkrementalny (obrotowy)
- Czerwona Dioda LED + 1x rezystor 1k Ohm
- Przycisk + 1x rezystor 1k Ohm

1.1.1 Arduino Uno

Arduino Uno jest wyposażone w mikrokontroler ATmega328P, który jest odpowiedzialny za wykonywanie programu zapisanego na płytce. Mikrokontroler ten obsługuje wiele funkcji, takich jak odczytywanie i zapisywanie stanu pinów cyfrowych oraz przetwarzanie sygnałów analogowych. Ma 14 cyfrowych pinów wejścia/wyjścia, z których 6 może być używanych jako wyjścia PWM do sterowania np. prędkością silników. Ponadto, posiada 6 pinów wejścia analogowego, z których można odczytywać napięcia z czujników i innych urządzeń analogowych. Zasilane jest przez podłączenie do źródła zewnętrznego lub przez port USB komputera. Ma wbudowany układ konwertera USB-UART, który umożliwia komunikację z komputerem i programowanie płytki za pomocą Arduino IDE lub innych środowisk programistycznych.

1.1.2 Miernik temperatury DS18B20

Miernik temperatury DS18B20 to cyfrowy czujnik temperatury wykorzystujący interfejs OneWire. Komunikacja z czujnikiem odbywa się poprzez wysyłanie sygnałów sterujących przez magistralę OneWire, a czujnik przekazuje dane pomiarowe w formacie cyfrowym. Czujnik ma unikalny adres, co umożliwia podłączenie wielu czujników do jednej magistrali. DS18B20 mierzy temperaturę w zakresie od -55℃ do +125℃ z dokładnością 0,5℃. Dane odczytane przez mikrokontroler mogą być dalej przetwarzane lub wyświetlane na ekranie. Dwa rezystory o wartości 5k Ohm (razem 10 km Ohm) są wykorzystywane do dostosowania wartości napięcia lub prądu w celu zapewnienia prawidłowego działania podłączonych elementów elektronicznych lub do stabilizacji sygnałów wejściowych.

1.1.3 Wyświetlacz LCD 20x4

Wyświetlacz LCD 20x4 to duży wyświetlacz alfanumeryczny o rozmiarze 20 kolumn na 4 wiersze. Komunikuje się z mikrokontrolerem za pomocą protokołu, zazwyczaj interfejsu równoległego, który wymaga wielu pinów do przesyłania danych. Podstawowe kroki komunikacji: - Przed rozpoczęciem komunikacji, należy skonfigurować pinout mikrokontrolera, aby przyporządkować odpowiednie piny do linii danych (D0-D7) oraz linii sterujących (np. RS - wybór trybu, E - sygnał zegara).

- Przesyłanie komend i danych: Do wyświetlania tekstu, sterowanie wyświetlaczem odbywa się poprzez przesyłanie komend i danych. Komendy są wysyłane w celu ustawiania trybu, czyszczenia ekranu, itp. Dane są przesyłane w celu wyświetlania konkretnych znaków lub napisów.

- Sygnały sterujące: W trakcie komunikacji, należy odpowiednio ustawiać sygnały sterujące, takie jak RS (Register Select) - informujący o przesyłaniu komendy lub danych, oraz E (Enable) - sygnał zegara inicjujący odczyt danych.

1.1.4 Enkoder inkrementalny (obrotowy)

Enkoder inkrementalny jest urządzeniem elektronicznym, które generuje impulsy w odpowiedzi na ruch obrotowy wału lub osi. Sygnały impulsowe pozwalają określić kierunek i liczbę obrotów, natomiast sygnał referencyjny wskazuje początek lub koniec pełnego obrotu. Enkoder inkrementalny jest podłączany do mikrokontrolera lub układu odbiorczego, który przetwarza sygnały i umożliwia monitorowanie prędkości, kierunku ruchu oraz określanie aktualnego położenia wału lub osi.

Pomiar zbocza opadającego w enkoderze polega na monitorowaniu zmiany stanu sygnału enkodera w momencie, gdy przechodzi on z wysokiego stanu logicznego na niski. Arduino może odczytywać to zbocze poprzez konfigurację odpowiednich pinów cyfrowych jako wejścia i korzystanie z przerwań (interrupts), które reagują na zmiany stanu sygnału. Wykorzystanie pomiaru zbocza opadającego umożliwia dokładne śledzenie ruchu enkodera i precyzyjne określanie liczby impulsów oraz kierunku obrotu.

1.1.5 Czerwona Dioda LED

W Arduino, czerwona dioda LED jest podłączana do odpowiedniego pinu cyfrowego na płytce. Po podłączeniu, można kontrolować diodę poprzez manipulację stanem tego pinu cyfrowego. Jest półprzewodnikowym komponentem elektronicznym, który emituje światło widzialne o dominującej długości fali w zakresie czerwonym. Działa na zasadzie zjawiska elektroluminescencji, gdzie przepływ prądu przez diodę powoduje reemisję energii w postaci światła. Składa się z dwóch warstw półprzewodnikowych, zwanymi anodą i katodą, które są połączone w prosty sposób. Podłączenie diody LED do źródła zasilania w odpowiedni sposób (zachowując polaryzację) powoduje przepływ prądu i emisję światła czerwonego przez diodę.

1.1.6 Przycisk

Przycisk składa się z dwóch styków, które są fizycznie połączone w momencie naciśnięcia przycisku i rozłączone po zwolnieniu. W Arduino, przycisk jest podłączany do jednego z pinów cyfrowych. Pin jest ustawiany jako wejście i odczytuje stan przycisku - wysoki (HIGH) lub niski (LOW). Aby odczytać stan przycisku w Arduino, wykorzystuje się funkcję digitalRead(), która sprawdza, czy pin cyfrowy odczytuje wartość HIGH lub LOW. W momencie naciśnięcia przycisku, pin odczytuje wartość HIGH, a po zwolnieniu wartość LOW.

1.2 Schemat podłączenia

1.2.1 Wyświetlacz LCD (Piny licząc od lewej):

1. GND - masa
2. Vcc - zasilanie dodatnie, 5V
3. V0 - regulacja kontrastu
4. RS - wybór rejestrów (komenda, dane)
5. RW - wybór opcji odczyt/zapis
6. E - zezwolenie na zapis do rejestrów
7. D0 - dane
8. D1 - dane
9. D2 - dane
10. D3 - dane
11. D4 - dane (używane)
12. D5- dane (używane)
13. D6 - dane (używane)

14. D7 - dane (używane)
15. Vpod - zasilanie dodatnie podświetlenia
16. GNDpod - masa podświetlenia

1.2.2 Miernik temperatury DS18B20 (Piny licząc od lewej):

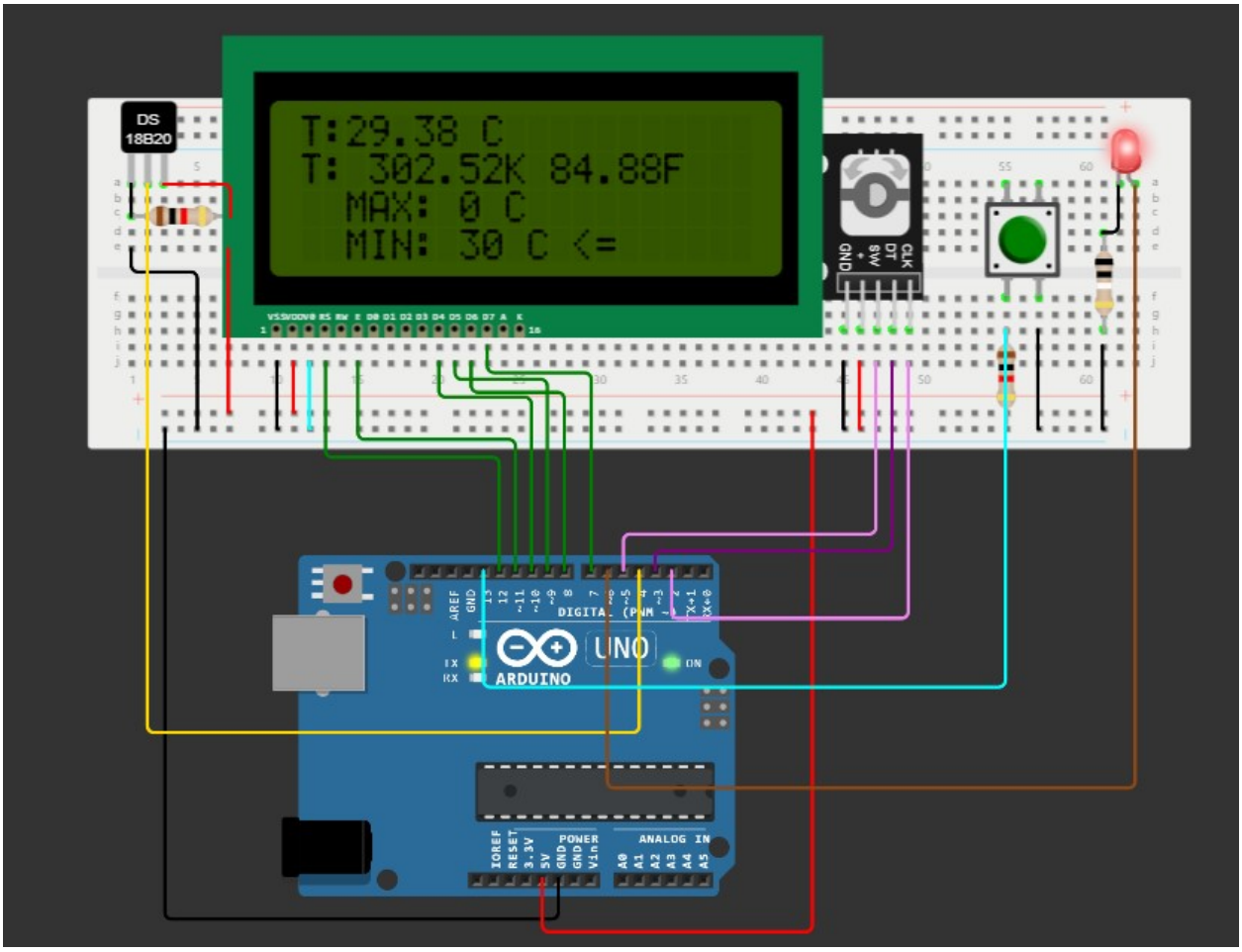
1. GND masa
2. PIN sygnałowy
3. Vcc Zasilanie 5V

1.2.3 Dioda LED

1. Anoda (Dłuższa nóżka) - do GND - masy przez rezystor
2. Katoda (Krótsza nóżka) - PIN sygnałowy

1.2.4 Enkoder (Piny licząc od lewej):

1. GND - masa
2. Vcc - zasilanie dodatnie 5v
3. SW - przycisk środkowy, PIN sygnałowy
4. SIB- PIN sygnałowy
5. SIA - PIN sygnałowy



Rysunek 1: Schemat podłączenia

1.3 Wykorzystane oprogramowanie

- Wokwi - symulator online
- Arduino IDE 2.1.0

1.4 Wykorzystane biblioteki zewnętrzne

- Standardowa biblioteka Arduino
- LiquidCrystal
- OneWire
- DallasTemperature

2. Problematyka

2.1 Problem układu Raspberry Pi Pico W - RP2040

Wybór Arduino Uno zamiast Rasberry PI Pico W rozwiązał problem nie działającej obsługi czunika DS18B20. Biblioteka 'OneWire' powodowała błędy podczas wgrywania programu na układ Pico. Po wielu nieudanych próbach udało się znaleźć odpowiednią obługę w MicroPython, jednak reszta założeń projektu tak jak obługa ekranu LCD, była dość złożona i powodowała problemy. Płytka Arduino Uno, lepiej współpracowało ze środowiskiem Arduino IDEE, i czas wgrywania oraz komplilacji był o wiele szybszy, co umożliwiło bardziej komfortową pracę.

Kod MicroPython do obsługi Ds18B20:

```
import machine, onewire, ds18x20, time

ds_pin = machine.Pin(21)

ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
roms = ds_sensor.scan()

def Temperatura(roms,ds_sensor):
  ds_sensor.convert_temp()
  time.sleep_ms(750)
  for rom in roms:
    temperatura = ds_sensor.read_temp(rom)
    print(f"Temperatura: {temperatura} C")
  time.sleep(2)

while True:
    Temperatura(roms,ds_sensor)
```

2.2 OOP - Programowanie obiektowe

Dodatkowo w celu ułatwienia i utrzymania przejrzystości kodu wykorzystane zostało programowanie obiektowe i podział na klasy i obiekty w C++.

2.3 Podział plików

Wydzielenie różnych funkcjonalności do osobnych plików, które: - ułatwiają rozbudowę - ułatwiają znalezienie błędu - są skalowanlne, pliki można wykorzystać wielokrotnie w innym projekcie

2.4 Testowanie

W głównej klasie 'MiernikTemperatury9' znajduję się metoda "Testing()" w której można wywołać funkcję do testowania czy podłączony element komunikuje się z Arduino.
Poprzez wywołanie na obiekcie metody **.test()**
Przykład testowania czujnika:

```
void Testing()
 {
   //Limit.detect(Btn.detectPress());
   // Serial.println("Test - arduino is connect");
   TemperatureSensor.test();
   // Lcd.test();
   // Btn.test();
   // Led.test();
 }

 void loop()
 {
   //Main();
   Testing();
 }
```

2.5 Problem jednoczesnego pomiaru temperatury i działania enkodera

Rozwiązanie problemu polegało na wprowadzeniu dodatkowej zmiennej readTemperatureFlag oraz pola previousTemperatura w klasie MiernikTemperatury9.

Zmienna readTemperatureFlag jest flagą, która określa, czy odczyt temperatury powinien zostać wykonany. Jeśli enkoder jest nieużywany, flaga zostaje ustawiona na true, co oznacza, że temperatura powinna zostać odczytana z czujnika. Natomiast jeśli enkoder jest używany, flaga jest ustawiana na false, a wtedy zamiast odczytywać temperaturę, wykorzystywana jest poprzednia zapisana wartość.

Pole previousTemperatura przechowuje poprzednią odczytaną temperaturę. W momencie, gdy flaga readTemperatureFlag jest ustawiona na true, oznacza to, że odczyt temperatury powinien zostać wykonany, więc aktualna temperatura jest zapisywana do pola previousTemperatura.

W funkcji Main(), jeśli enkoder jest nieużywany (stan LOW na pinie ENCODER_A_PIN), flaga readTemperatureFlag jest ustawiana na true. Wówczas wywoływana jest metoda text_temperature() z obiektu TemperatureSensor, a następnie wynik jest konwertowany na liczbę całkowitą (int) za pomocą metody toInt(). Wartość ta jest przypisywana do pola Temperatura, a także jest wyświetlana na ekranie LCD.

Natomiast jeśli enkoder jest używany, flaga readTemperatureFlag jest ustawiana na false, a wówczas wywoływana jest metoda Testing() w celu przeprowadzenia innych operacji (np. testowanie przycisków czy diod LED). Aktualna temperatura nie jest odczytywana, a na ekranie LCD wyświetlana jest poprzednia zapisana wartość z pola previousTemperatura.

W przypadku zmiany limitu temperatury (przez wciśnięcie przycisku), odpowiedni limit i poprzednia temperatura są przekazywane do metody limit_info() obiektu Led.

W ten sposób zastosowanie flagi readTemperatureFlag oraz pola previousTemperatura pozwala na wykrywanie czy enkoder jest używany i umożliwia zachowanie poprzedniej odczytanej temperatury, gdy enkoder jest w użyciu.

3. Pliki źródłowe projektu

- main.ino
- CONFIG.cpp
- ds18b20.cpp
- ScreenLcd.cpp
- LimitTemperature.cpp
- DiodeLed.cpp
- Button.cpp

3.1 Inicjalizacja - main.ino

Plik main.ino zawiera klasę **MiernikTemperatury9**, która agreguje cały kod i wywołuje wszystkie metody z innych klas.

```
#include "CONFIG.cpp"

class MiernikTemperatury9
{
private:
  TemperatureSensorDS18B20 TemperatureSensor;
  ScreenLcd Lcd;
  DiodeLed Led;
  Button Btn;
  LimitTemperature Limit;
  float Temperatura;
  float previousTemperatura;
  bool readTemperatureFlag = true;

public:
  void setTemperatura(float newT) { Temperatura = newT; }
  float getTemperatura() { return Temperatura; }

  void Testing()
  {
    Limit.detect(Btn.detectPress());
    // Serial.println("Test - arduino is connect");
    // TemperatureSensor.test();
    // Lcd.test();
    // Btn.test();
    // Led.test();
  }

  void loop()
  {
    Main();
    // Testing();
  }
  void setup()
  {
    Serial.begin(SERIAL_PORT);
    TemperatureSensor.setup();
    Lcd.setup();
  }
```

```
  MiernikTemperatury9() :
    TemperatureSensor(DS_PIN),
    Lcd(RS, E, D4, D5, D6, D7),
    Led(LED_PIN),
    Btn(BTN_PIN){}

  void Main()
  {
    int choose = Btn.detectPress();

    String text[3];
    Limit.detect(Btn.detectPress());
    text[1] = Limit.getTextMin();
    text[0] = Limit.getTextMax();

    if (digitalRead(ENCODER_A_PIN) == LOW) {
      readTemperatureFlag = true;
      setTemperatura(TemperatureSensor.readTemperature());
      text[2] = getTemperatura();

      if (choose == 1) Led.limit_info(Limit.getMax(), text[2].toInt());
      if (choose == 0) Led.limit_info(Limit.getMin(), text[2].toInt());


    } else {
      readTemperatureFlag = false;
      Testing();
      text[2] = previousTemperatura;

      if (choose == 1) Led.limit_info(Limit.getMax(), text[2].toInt());
      if (choose == 0) Led.limit_info(Limit.getMin(), text[2].toInt());

    }

    Lcd.displayTextLcd(text);

    if (readTemperatureFlag) {
      previousTemperatura = Temperatura;
    }

  }
};

MiernikTemperatury9 projekt;
void setup() { projekt.setup(); }
void loop() { projekt.loop(); }
```

3.2 Plik konfiguracyjnych - CONFIG.cpp

Zawiera konfigurację wszystkich pinów oraz załącza wszystkie klasy.

```
// DEPENDENCES
#include "ScreenLcd.cpp"
#include "DiodeLed.cpp"
```

```
#include "MyEncoder.cpp"
#include "Button.cpp"
#include "ds18b20.cpp"

#define SERIAL_PORT 9600

// DS18B20 TEMPERATURE PIN
#define DS_PIN  13

// LCD SCREEN PIN
#define RS 12
#define E 11
#define D4 10
#define D5 9
#define D6 8
#define D7 7

// COUNTER ENCODER PIN
#define ENCODER_A_PIN 2
#define ENCODER_B_PIN 3
#define RESET_BTN_PIN 5

// LED DIODE PIN
#define LED_PIN 6

// MIN/MAX BUTTON PIN
#define BTN_PIN 4
```

3.3 Wyświetlacz LCD 20x4 - ScreenLcd.cpp

```
#include <LiquidCrystal.h>
#include <Arduino.h>
#define LCD_CHAR 20
#define LCD_ROWS 4

class ScreenLcd
{
  private:
    LiquidCrystal lcd;

  public:
    ScreenLcd(int rs, int en, int d4, int d5, int d6, int d7)
      : lcd(rs, en, d4, d5, d6, d7){}

    void setup() {
      lcd.begin(LCD_CHAR, LCD_ROWS);
    }

    void displayText(const String line) {
      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print(line);
    }

    void displayTextLcd(const String line[2]){
      lcd.clear();

      lcd.setCursor(0, 0);
      lcd.print(line[0]);

      lcd.setCursor(0, 1);
      lcd.print(line[1]);

      lcd.setCursor(0, 2);
      lcd.print(line[2] + " C");
    }

    void test() {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("test");
      delay(2000);
    }
};
```

3.4 Miernik temperatury DS18B20 - ds18b20.cpp

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Arduino.h>
class TemperatureSensorDS18B20
{
  private:
    OneWire oneWire;
    DallasTemperature sensor;
    unsigned long lastReadTime;
    unsigned long readInterval;

  public:
    TemperatureSensorDS18B20(int pin, unsigned long interval = 1000) : oneWire(pin), sensor(&oneWire), 

    void setup() { sensor.begin();}

    float readTemperature() {
      unsigned long currentTime = millis();
      if (currentTime - lastReadTime >= readInterval) {
        sensor.requestTemperatures();
        lastReadTime = currentTime;
      }
      return sensor.getTempCByIndex(0);
    }

    float readKelvin() { return readTemperature() + 273.15;}

    float readFahrenheit() {
      float c = readTemperature();
      float f = (c * 9.0 / 5.0) + 32.0;
      return f;
    }

    String text_temperature() { return "T:" + String(readTemperature()) + " C"; }
    String text_kelvin() { return String(readKelvin()) + "K"; }
    String text_fahrenheit() { return String(readFahrenheit()) + "F"; }
    String text_second_row() { return "T: " + text_kelvin() + " " + text_fahrenheit(); }

    void test() {
        float get_temperature = readTemperature();
        String text_temperature = "T: " + String(get_temperature) + " C";
        Serial.println(text_temperature);
    }
};
```

3.5 Enkoder - LimitTemperature.cpp

```
#include <Arduino.h>
#define ENCODER_A_PIN 2
#define ENCODER_B_PIN 3
#define ENCODER_RESET_PIN 5

class LimitTemperature
{
private:
  int Min;
  int Max;
  int SIA;
  int SIB;
  int SI_SW;
  int a = 1;
  int b = 1;
  int t = 0;

public:
  String text_max = "0";
  String text_min = "0";

  LimitTemperature() : Min(0), Max(0), SIA(ENCODER_A_PIN), SIB(ENCODER_B_PIN), SI_SW(ENCODER_RESET_PIN)
  void setup()
  {
    pinMode(SIB, INPUT);
    pinMode(SIA, INPUT);
  }

  void setMin(int newMin) { Min = newMin; }
  void setMax(int newMax) { Max = newMax; }

  int getMin() { return Min; }
  int getMax() { return Max; }

  void setTextMax(String newtext) { text_max = newtext; }
  void setTextMin(String newtext) { text_min = newtext; }

  String getTextMax() { return text_max;}
  String getTextMin() { return text_min;}


  void testPrint(String cursor_max, String cursor_min)
  {
      Serial.println("===========");
      Serial.println("MAX: " + String(getMax()) + cursor_max);
      Serial.println("MIN: " + String(getMin()) + cursor_min);
      Serial.println("===========");
  }
  void testPrintMax() { testPrint(" <==", " "); }
  void testPrintMin() { testPrint(" ", " <=="); }

  void detect(int detect)
```

```
  {
    // LIMIT MAX
    int choose = detect;
    if (choose == 1){
        a = digitalRead(SIA);

        // reset counter
        if (digitalRead(SI_SW) == LOW) {setMax(0); testPrintMax();}

        // enkdoer
        if ((b == 1) && (a == 0) && ((millis() - t) > 50))
        {
          t = millis();
          (digitalRead(SIB) == 1) ? setMax(getMax() + 1) : setMax(getMax() - 1);
          testPrintMax();
        }
        setTextMax("MAX: " + String(getMax()) + " C <==");
        setTextMin("MIN: " + String(getMin()) + " C");
    }

    // LIMIT MIN
    if (choose == 0)
    {
      // reset counter
      if (digitalRead(SI_SW) == LOW) { setMin(0); testPrintMin(); }

      // enkdoer
      a = digitalRead(SIA);
      if ((b == 1) && (a == 0) && ((millis() - t) > 50))
      {
        t = millis();
        (digitalRead(SIB) == 1) ? setMin(getMin() + 1) : setMin(getMin() - 1);
        testPrintMin();
      }
        setTextMax("MAX: " + String(getMax()) + " C");
        setTextMin("MIN: " + String(getMin()) + " C <==");
    }
  }
};
```

3.6 Przycisk - Button.cpp

```
#include <Arduino.h>
class Button
{
private:
  int pin;
  int buttonState;
  int prevButtonState;
  unsigned long lastDebounceTime;
  bool isButtonOn;
public:
  Button(int pin) : pin(pin), buttonState(HIGH), prevButtonState(HIGH), lastDebounceTime(0), isButtonOn
    pinMode(pin, INPUT);
  }
  void update() {
    int reading = digitalRead(pin);

    if (reading != prevButtonState) {
      lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > 50) {
      if (reading != buttonState) {
        buttonState = reading;

        if (buttonState == LOW) {
          isButtonOn = !isButtonOn;

          if (isButtonOn) {
            Serial.println("Button: MAX");
          } else {
            Serial.println("Button: MIN");
          }
        }
      }
    }
    prevButtonState = reading;
  }

  bool isOn() { return isButtonOn;}
  int detectPress(){
    update();
    if(isOn()) { return 1; }
    else { return 0;}
  }
  void test(){
    int choose = detectPress();
    if(choose==1) Serial.println("on");
    if(choose==0) Serial.println("off");
  }
};
```

3.7 Dioda LED - DiodeLed.cpp

```
#include <Arduino.h>
class DiodeLed
{
  private:
    int ledPin;

  public:
    DiodeLed(int pin) {
      ledPin = pin;
      pinMode(ledPin, OUTPUT);
    }

    void on(){
      digitalWrite(ledPin, HIGH);
    }

    void off(){
      digitalWrite(ledPin, LOW);
    }

    void blink() {
      on();
      delay(1000);

      off();
      delay(1000);
    }

    void limit_info(long limit, int temperature)
    {
      if(temperature > limit) on();
      else off();
    }

    void test()
    {
      on();
    }
};
```