

Wydział Nauk Technicznych i Ekonomicznych

Informatyka

Techniki mikroprocesorowe i systemy wbudowane

Adrian Rudnicki

**Bezprzewodowy sterownik do pieca centralnego ogrzewania**

Napisane pod kierunkiem:

mgr inż. Jan Duda

Legnica 2022

Spis treści

[1. Przeznaczenie urządzenia i jego właściwości 2](#_Toc124106470)

[2. Elementy potrzebne do budowy urządzenia 2](#_Toc124106471)

[3. Schemat urządzenia 6](#_Toc124106472)

[3.1. Schemat nadajnika 6](#_Toc124106473)

[3.2. Schemat odbiornika 6](#_Toc124106474)

[4. Program działania 7](#_Toc124106475)

[4.1. Kod programu nadajnika 7](#_Toc124106476)

[4.2. Kod programu odbiornika 12](#_Toc124106477)

[5. Ocena zastosowanego rozwiązania, możliwości udoskonalenia 14](#_Toc124106478)

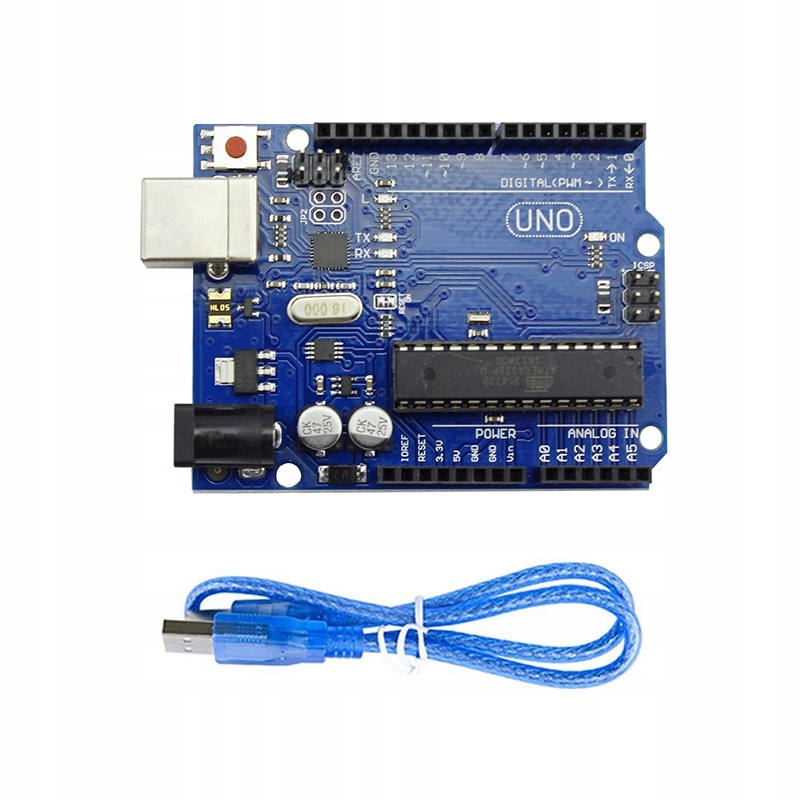
[Spis rysunków 15](#_Toc124106479)

1. Przeznaczenie urządzenia i jego właściwości

Urządzenie, które zbudowałem służy do sterowania piecem centralnegoogrzewania. Jest to termoregulator bezprzewodowy, który składa się z dwóchgłównych modułów. Moduł pomiaru i sterowania (dalej nadajnik) oraz modułwykonawczy (dalej odbiornik). Dzięki zastosowaniu połączenia bezprzewodowegomożemy przenosić nadajnik między różnymi pomieszczeniami, w których planujemypomiar temperatury otoczenia w celu osiągnięcia temperatury docelowej.Zastosowane przy nadajniku LCD oraz klawisze pozwala naprostą zmianędocelowej temperatury oraz na wyświetlanie temperatury aktualnej. Odbiornik maprostszą budowę, ponieważ przekazuje on sygnał załączenia do pieca centralnegoogrzewania przez odpowiednie wysterowanie modułu przekaźnika.

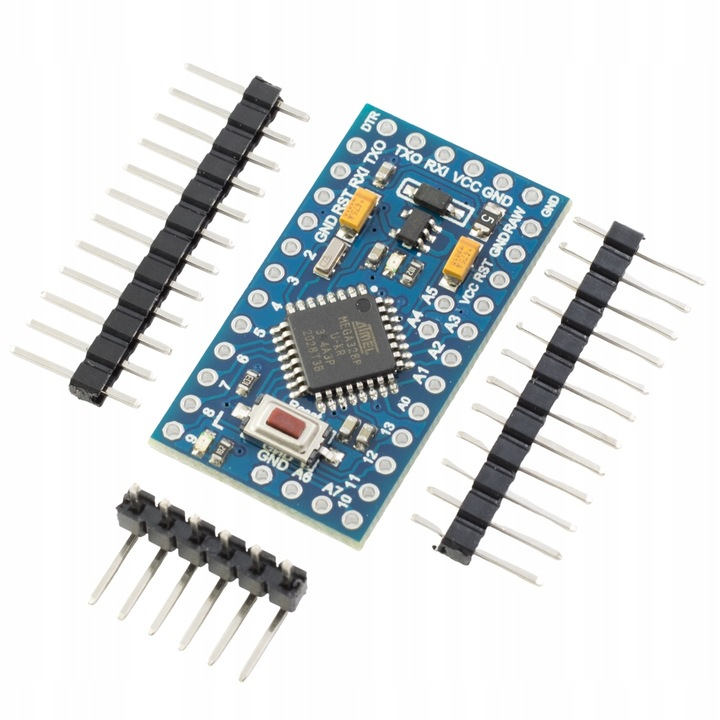
1. Elementy potrzebne do budowy urządzenia

**Arduino UNO** najpopularniejsza płytka do nauki elektroniki i programowania, która posiada 14 uniwersalnych pinów GPIO (wejść/wyjść) i 6 wejść analogowych. Na płytce znajdziemy 8-bitowy mikrokontroler AVR ATmega328P, którego piny (w większości) zostały wyprowadzone na, charakterystyczne dla Arduino, złącza dla shieldów, czyli nakładek rozszerzających możliwości płytki. Na płytce znajduje się również drugi mikrokontroler (ATmega16U2), który pozwala na komunikację z komputerem oraz na wgrywanie programów.



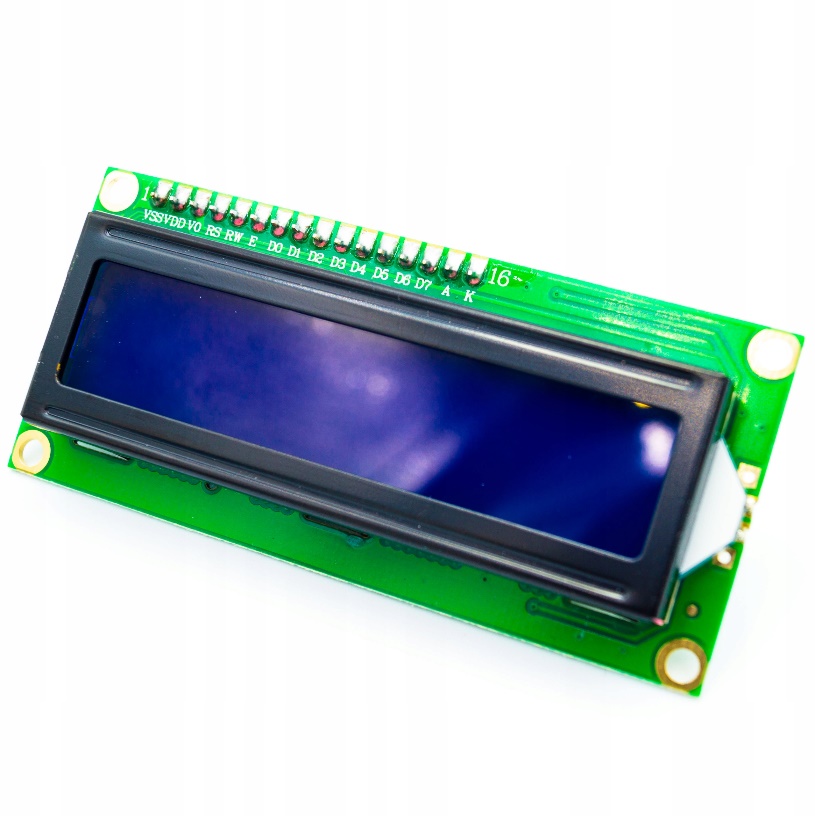
*rys. 1 Arduino UNO*

**Arduino Pro Mini** płytka (podobna do Arduino UNO) ale dużo mniejsza, przydatna do wykorzystania w finalnych projektach, gdy jesteśmy ograniczeni dostępnym miejscem. Aby ją zaprogramować należy wykorzystać zewnętrzny programator, ponieważ na płytce, tak jak w przypadku Arduino UNO, nie znajduje się układ do komunikacji z komputerem. Na płytce znajdziemy 8-bitowy mikrokontroler AVR ATmega328P, która posiada 14 uniwersalnych pinów GPIO (wejść/wyjść) i 6 wejść analogowych.



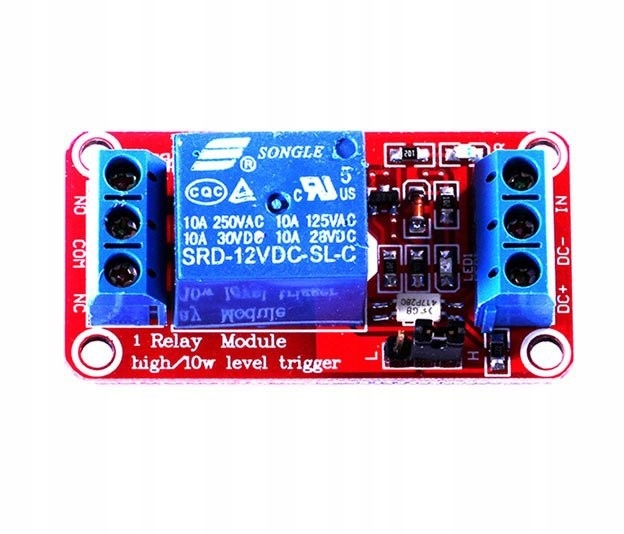
*rys. 2 Adruino Pro Mini*

**Wyświetlacz LCD I2C 16x2** charakteryzuje go niebieskie podświetlenie tła i białe litery tekstu. Sterowanie odbywa się przy pomocy biblioteki LiquidCrystal\_I2C.h. Dzięki zastosowaniu magistrali I2C między Arduino a wyświetlaczem zaoszczędamy sporo wejść/wyjść mikrokontrolera, które możemy wykorzystać w dalszej rozbudowie.



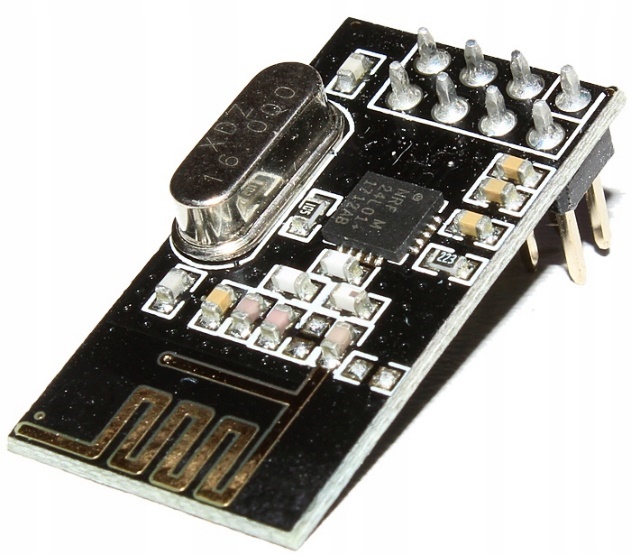
*rys. 3 Wyświetlacz LCD*

**Moduł przekaźnika** przy jego pomocy możemy załączać urządzenia wymagające dużej mocy, możemy łączyć obwody sterowania pracujące w wyższym napięciu. Przy wykorzystaniu takiego modułu możemy określić rodzaj sygnału załączenia przekaźnika (wysoki lub niski).



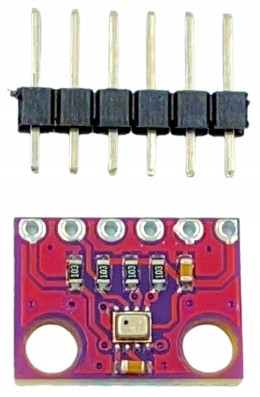
*rys. 4 Moduł przekaźnika*

**NRF24L01** moduł radiowy (nadajnik i odbiornik) działający w paśmie 2,4 GHz komunikujący się z mikrokontolerem przez magistralę SPI. Za pomocą tego modułu NRF24L01 możemy wysłać wiadomość do konkretnego odbiorcy. Możemy odebrać wiadomość od jakiegoś konkretnego nadawcy. W niektórych szczególnych zastosowaniach należy również wykonywać przełączanie pomiędzy stanem odbioru i nadawania.



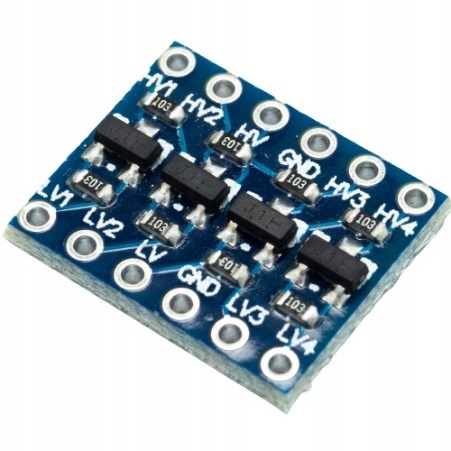
*rys. 5 Moduł komunikacji bezprzewodowej NRF24L01*

**BMP280** cyfrowy czujnik ciśnienia z dokładnością 1 hPa. Posiada wbudowany czujnik temperatury (wykorzystywany do ustalenia aktualnej temperatury). Układ komunikuje się z Arduino za pomocą magistrali I2C. Czujnik pozwala na pomiar temperatury od -40°C do +85°C, a jego dokładność pomiaru to +/- 1°C.



*rys. 6 Cyfrowy czujnik ciśnienia i temperatury BMP280*

**Dwukierunkowy konwerter stanów logicznych** pozwala na łączenie urządzeń logicznych o różnych napięciach bez ryzyka uszkodzenia urządzenia.



*rys. 7 Dwukierunkowy konwerter stanów logicznych*

**Przetwornica napięcia step-down** oparta na LM2596S przy pomocy tego modułu, można płynnie regulować napięcie wyjściowe w zakresie od 1,5V do 35V prądu stałego przez zmianę rezystancji potencjometru, na płytce modułu. Moduł nie potrafi podnosić wartości napięcia, więc ważne jest, aby napięcie zasilania takiej przetwornicy było wyższe niż te, które jest wymagane na wyjściu.

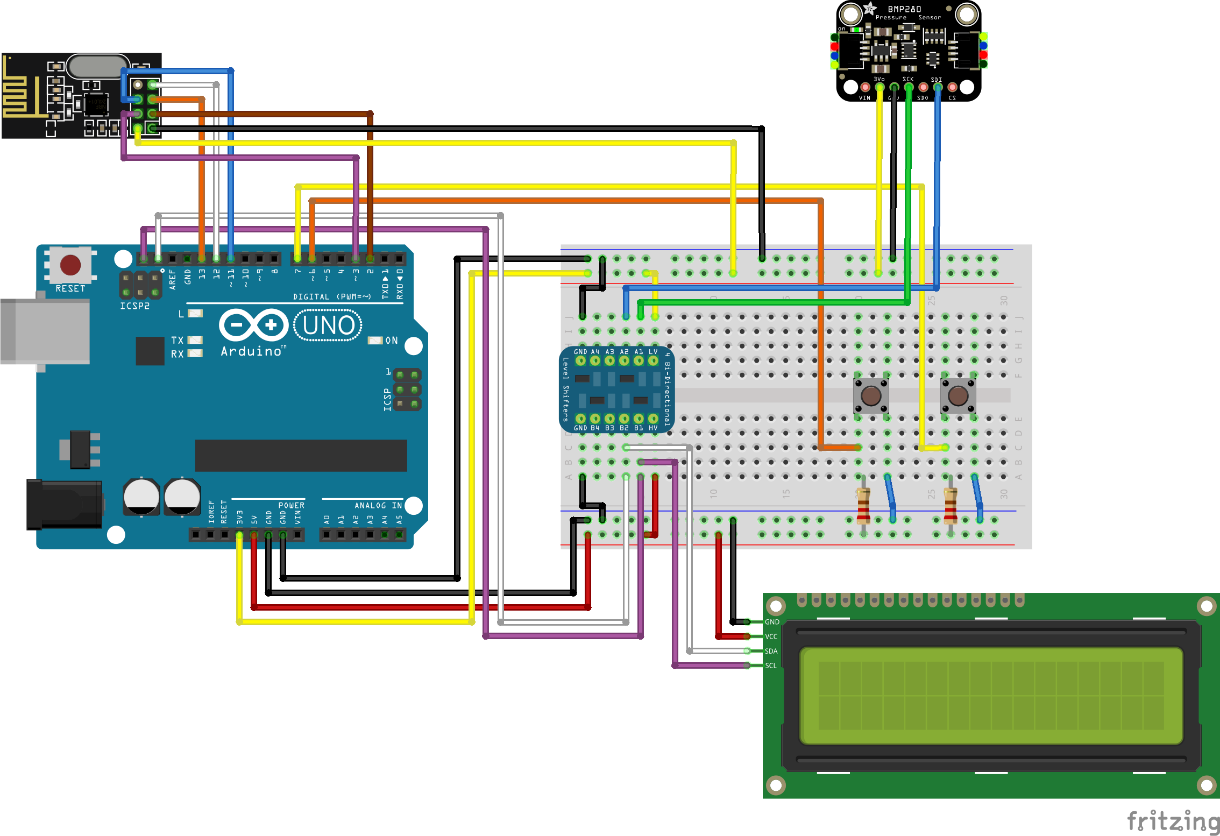


*rys. 8 Przetwornica napięcia*

1. Schemat urządzenia

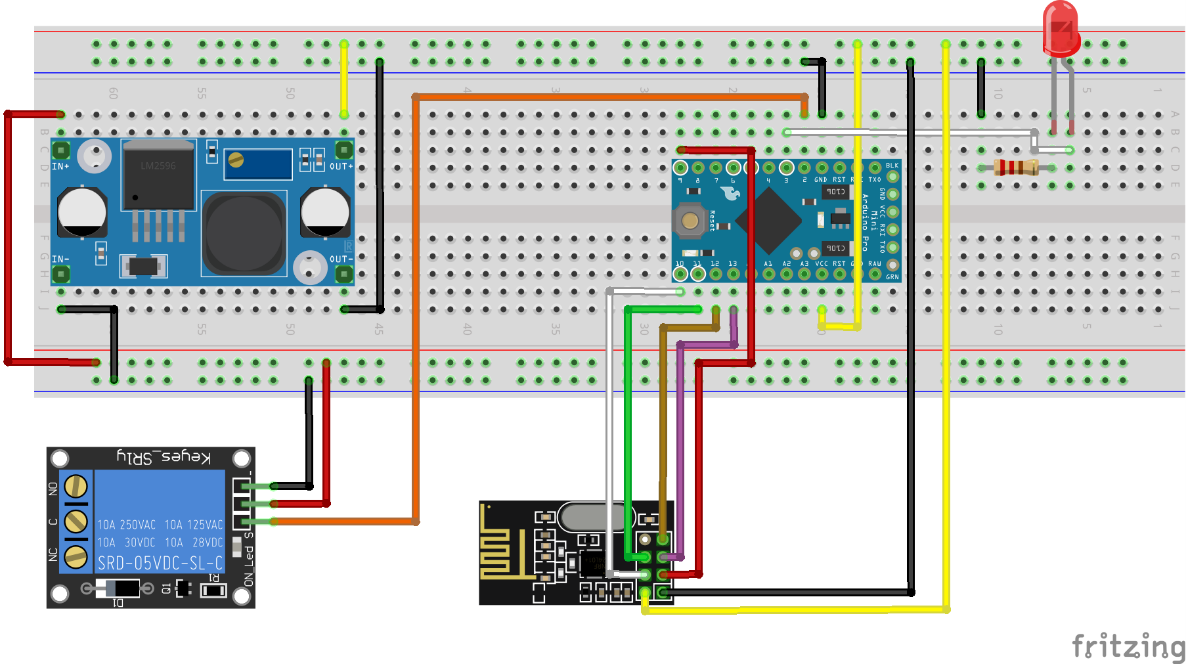
Schematy urządzenia zostały utworzone w taki sposób, aby każdy mógł je zrozumieć. Zastosowane grafiki modułów są bardzo czytelne i zostały połączone kolorowymi liniami z odpowiednimi punktami. Schematy są opracowane w programie Fritzing.

* 1. Schemat nadajnika



*rys. 9 Schemat połączeń nadajnika*

* 1. Schemat odbiornika



*rys. 10 Schemat połączeń odbiornika*

Schematy połączeń są dołączone jako załącznik do tego dokumentu. Są w wyższej jakości oraz pozwolą na zaimportowanie schematu do programu Fritzing

1. Program działania
   1. Kod programu nadajnika

#include <Wire.h>

#include <SPI.h>

#include <Adafruit\_BMP280.h>

#include <LiquidCrystal\_I2C.h>

#include "NRFLite.h"

// Configure static variable

const static uint8\_t RADIO\_ID = 0;

const static uint8\_t DESTINATION\_RADIO\_ID = 1;

const static uint8\_t PIN\_RADIO\_CE = 2;

const static uint8\_t PIN\_RADIO\_CSN = 3;

const static uint8\_t BTN\_UP = 6;

const static uint8\_t BTN\_DOWN = 7;

const static uint8\_t LCD\_ADDRESS = 0x27;

const static uint8\_t TEMP\_SENSOR\_ADDRESS = 0x76;

const static float HYSTERESIS\_TRESHOLD = 0.2;

const static float TEMP\_SET\_STEP = 0.2;

// Define special character for lcd

byte \_heating[8] = {

  B00000,

  B00100,

  B00100,

  B00110,

  B01111,

  B11111,

  B11111,

  B01110

};

// Using for send command data

enum RadioPacketType {

  TurnOn,

  TurnOff,

};

struct RadioPacket {

  RadioPacketType PacketType;

  uint8\_t FromRadioId;

  uint32\_t OnTimeMillis;

};

// Global variables and objects

NRFLite \_radio;

LiquidCrystal\_I2C \_lcd(LCD\_ADDRESS, 16, 2);

Adafruit\_BMP280 bmp;

uint32\_t \_lastRelayStateSendTime;

uint32\_t \_lastLcdUpdate;

float \_setTemperature = 23.0;

bool \_relayState = false;

uint8\_t \_communicationError = 0;

void setup() {

  Serial.begin(9600);

  pinMode(BTN\_UP, INPUT);

  pinMode(BTN\_DOWN, INPUT);

  // Communicate with NRF24L01

  if (!\_radio.init(RADIO\_ID, PIN\_RADIO\_CE, PIN\_RADIO\_CSN)) {

    Serial.println("Blad polaczenia z radiem");

    while (1);

  }

  // Comunicate with BMP280

  if (!bmp.begin(TEMP\_SENSOR\_ADDRESS)) {

    Serial.println("Problem z polaczeniem termometru");

    while (1);

  }

  // Initialize lcd

  \_lcd.init();

  \_lcd.backlight();

  \_lcd.createChar(0, \_heating);

  \_lcd.setCursor(0, 0);

  lcdUpdate();

}

void loop() {

  // check if keybord key is pressed

  switch (checkKeyboard()) {

    case 1:

      // Set temperature higher

      setTemperature(TEMP\_SET\_STEP);

      break;

    case 2:

      // Set temperature lower

      setTemperature(TEMP\_SET\_STEP \* -1.0);

      break;

    default:

      break;

  }

  // Send a command for relay state every 10 seconds.

  if (millis() - \_lastRelayStateSendTime > 9999) {

    checkState();

    if (\_relayState) {

      sendTurnOn();

    } else {

      sendTurnOff();

    }

  }

  // Show any received data.

  checkRadio();

  // Check if is communication problem

  if (\_communicationError > 6) {

    showCommunicationError(\_communicationError);

  } else {

    if (millis() - \_lastLcdUpdate > 999) {

      lcdUpdate();

      \_lastLcdUpdate = millis();

    }

  }

}

// Send turn on state to reciver

void sendTurnOn() {

  Serial.println("Wysylanie wiadomosci wlacz...");

  RadioPacket radioData;

  radioData.PacketType = TurnOn;

  radioData.FromRadioId = RADIO\_ID;

  radioData.OnTimeMillis = millis();

  if (\_radio.send(DESTINATION\_RADIO\_ID, &radioData, sizeof(radioData)))

  {

    Serial.println("Odbiornik odpowiedzial na wiadomosc");

    \_communicationError = 0;

  } else {

    \_communicationError++;

    Serial.println("Odbiornik nie odpowiedzial na wiadomosc");

  }

  \_lastRelayStateSendTime = millis();

}

// Send turn off state to reciver

void sendTurnOff() {

  Serial.println("Wysylanie wiadomosci wylacz...");

  RadioPacket radioData;

  radioData.PacketType = TurnOff;

  radioData.FromRadioId = RADIO\_ID;

  radioData.OnTimeMillis = millis();

  if (\_radio.send(DESTINATION\_RADIO\_ID, &radioData, sizeof(radioData)))

  {

    Serial.println("Odbiornik odpowiedzial na wiadomosc");

    \_communicationError = 0;

  } else {

    \_communicationError++;

    Serial.println("Odbiornik nie odpowiedzial na wiadomosc");

  }

  \_lastRelayStateSendTime = millis();

}

void checkRadio() {

  while (\_radio.hasData())

  {

    RadioPacket radioData;

    \_radio.readData(&radioData);

    sendToSerial(radioData, "Odbiornik wyslal: ");

  }

}

void sendToSerial(RadioPacket radioData, String msg) {

  msg += radioData.FromRadioId;

  msg += ", ";

  msg += radioData.OnTimeMillis;

  msg += " ms";

  Serial.println(msg);

}

uint8\_t checkKeyboard() {

  delay(200);

  if (!digitalRead(BTN\_UP)) return 1;

  if (!digitalRead(BTN\_DOWN)) return 2;

  return 0;

}

// Check if is need to turn on or off heating

void checkState() {

  float actualTemp = bmp.readTemperature();

  float hysteresis[] = { \_setTemperature - HYSTERESIS\_TRESHOLD, \_setTemperature + HYSTERESIS\_TRESHOLD };

  if (actualTemp < hysteresis[1] && actualTemp > hysteresis[0]) \_relayState = \_relayState;

  else if (actualTemp < hysteresis[0]) \_relayState = true;

  else if (actualTemp > hysteresis[1]) \_relayState = false;

  lcdUpdate();

}

// Set temperature

void setTemperature(float temp) {

  if (\_setTemperature > 10.0 && \_setTemperature < 32.0)

    \_setTemperature = \_setTemperature + temp;

  lcdUpdate();

}

// Update lcd things

void lcdUpdate() {

  String actualTemp = "Act temp: ";

  String setTemp = "Set temp: ";

  actualTemp += String(bmp.readTemperature(), 1);

  setTemp += String(\_setTemperature, 1);

  \_lcd.clear();

  \_lcd.setCursor(0, 0);

  \_lcd.print(actualTemp);

  \_lcd.setCursor(0, 1);

  \_lcd.print(setTemp);

  if (\_relayState) {

    \_lcd.setCursor(15, 1);

    \_lcd.write(byte(0));

  }

}

void showCommunicationError(uint8\_t errors) {

  \_lcd.clear();

  \_lcd.setCursor(0, 0);

  \_lcd.print("Connection err");

  \_lcd.setCursor(0, 1);

  \_lcd.print("num of err ")

  \_lcd.print(errors);

}

* 1. Kod programu odbiornika

#include "SPI.h"

#include "NRFLite.h"

// Configuration static variable

const static uint8\_t RADIO\_ID = 1;

const static uint8\_t DESTINATION\_RADIO\_ID = 0;

const static uint8\_t PIN\_RADIO\_CE = 9;

const static uint8\_t PIN\_RADIO\_CSN = 10;

const static uint8\_t PIN\_RELAY = 2;

const static uint8\_t PIN\_LED\_ERROR = 3;

// Define RadioPacketType for determine what reciver need to do when will recive message from transmiter

enum RadioPacketType {

  TurnOn,

  TurnOff,

};

// Create struct for diffrent needed data

struct RadioPacket {

  RadioPacketType PacketType;

  uint8\_t FromRadioId;

  uint32\_t OnTimeMillis;

};

// Global variables and objects

NRFLite \_radio;

uint32\_t \_lastGetData;

void setup() {

  Serial.begin(9600);

  pinMode(PIN\_RELAY, OUTPUT);

  pinMode(PIN\_LED\_ERROR, OUTPUT);

  digitalWrite(PIN\_LED\_ERROR, HIGH);

  // Communicate with radio

  if (!\_radio.init(RADIO\_ID, PIN\_RADIO\_CE, PIN\_RADIO\_CSN)) {

    Serial.println("Problem polaczenia z radiem");

    digitalWrite(PIN\_LED\_ERROR, LOW);

    while (1);

  }

}

void loop() {

  // Show any received data.

  checkRadio();

  if(millis() - \_lastGetData > 59999) {

    setRelayState(false);

    digitalWrite(PIN\_LED\_ERROR, LOW);

    delay(500);

    digitalWrite(PIN\_LED\_ERROR, HIGH);

    delay(500);

  }

}

void checkRadio() {

  while (\_radio.hasData())  // 'hasData' puts the radio into Rx mode.

  {

    RadioPacket radioData;

    \_radio.readData(&radioData);

    if (radioData.PacketType == TurnOn) {

      sendToSerial(radioData, "Turn on from ");

      setRelayState(true);

    } else if (radioData.PacketType == TurnOff) {

      sendToSerial(radioData, "Turn of from ");

      setRelayState(false);

    }

    \_lastGetData = millis();

  }

}

// Serial for radioData

void sendToSerial(RadioPacket radioData, String msg) {

  msg += radioData.FromRadioId;

  msg += ", ";

  msg += radioData.OnTimeMillis;

  msg += " ms";

  Serial.println(msg);

}

// Set relay

void setRelayState(bool state) {

  if(state) digitalWrite(PIN\_RELAY, HIGH);

  else if(!state) digitalWrite(PIN\_RELAY, LOW);

}

1. Ocena zastosowanego rozwiązania, możliwości udoskonalenia

Urządzenie jest przydatne i w pełni funkcjonalne. Jednak nie mogę ukrywać, że posiada wady. Pierwszą z nich, która wygląda na najbardziej istotną, jest brak zasilania bateryjnego w module nadajnika. Moduł ten z natury będzie przenoszony z miejsca na miejsce, a takim wypadku zasilanie bateryjne jest wręcz niezbędne. Przy module odbiornika najczęściej mamy dostęp do stałego źródła zasilania. Kolejną wadą urządzenia jest brak możliwości programowania harmonogramu. Urządzenie udostępnia tylko możliwość ustawienia temperatury docelowej, do której będzie dążyć. Termoregulatory dostępne na rynku posiadają możliwość zaprogramowania harmonogramu uruchamiania pieca centralnego ogrzewania, czy wymaganą temperaturę w danych dniach o danych godzinach. Na szczęście moje urządzenie nie wymaga przebudowy elektronicznej, aby dodać taką funkcjonalność, wystarczy rozbudować oprogramowanie modułu nadajnika. W ramach rozbudowy urządzenia dobrym rozwiązaniem będzie zaprojektowanie obudowy na oba główne moduły. Można tutaj wykorzystać druk 3D. Kolejnym rozwinięciem projektu może być rozbudowa oprogramowania, w taki sposób, aby nadajnik usypiał się w celu zużywania mniejszej ilości energii. Wybudzenie mogłoby następować: co jakiś interwał czasowy, aby zmierzyć temperaturę otoczenia i podjąć odpowiednie akcje; oraz gdy użytkownik przystąpi do interakcji z interfejsem modułu nadajnika.

Spis rysunków

[*rys. 1 Arduino UNO* 2](#_Toc124106391)

[*rys. 2 Adruino Pro Mini* 3](#_Toc124106392)

[*rys. 3 Wyświetlacz LCD* 3](#_Toc124106393)

[*rys. 4 Moduł przekaźnika* 4](#_Toc124106394)

[*rys. 5 Moduł komunikacji bezprzewodowej NRF24L01* 4](#_Toc124106395)

[*rys. 6 Cyfrowy czujnik ciśnienia i temperatury BMP280* 5](#_Toc124106396)

[*rys. 7 Dwukierunkowy konwerter stanów logicznych* 5](#_Toc124106397)

[*rys. 8 Przetwornica napięcia* 5](#_Toc124106398)

[*rys. 9 Schemat połączeń nadajnika* 6](#_Toc124106399)

[*rys. 10 Schemat połączeń odbiornika* 6](#_Toc124106400)