Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr.5.1
Disciplina: IoT
Tema: Control funcțional - ON-OFF cu Histereza

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

1. Sa se realizeze o aplicatie in baza de MCU care va implementa sisteme de control pentru control temperatura sau umeditate cu aplicarea aplicarea metodei de control On-Off cu histeresis cu actionare prin releu.
2. Set point (valoarea de referinta pentru control) se va seta de la una din surse, la alegere
   - un potentiometru;
   - doua butoane pentru UP/Down;
   - sensor encoder;
   - keypad;
   - interfata serial
3. Valoarea de Setpoint si cea Curenta se vor afisa la LCD.

Obiective:

1. Afișarea valorilor pe LCD;
2. Asigurarea controlul funcțional al releului în funcție de temperatură;
3. Reutilizarea componentelor precedente.

INTRODUCERE

Actuatorii joacă un rol esențial în sistemele de control automatizat, servind ca elemente de acționare care transformă semnalele de control primite din partea sistemului în mișcare sau acțiuni fizice concrete. În contextul controlului temperaturii și umidității, actuatorii sunt responsabili pentru activarea și dezactivarea echipamentelor de răcire, încălzire sau umidificare în funcție de setările de control. Controlul de tip On-Off cu histerezis este o metodă de control simplă și eficientă, în care actuatorul este pornit sau oprit în funcție de abaterile valorii măsurate de la un punct de referință (setpoint), dar cu un interval (histerezis) care împiedică activarea frecventă și nejustificată.

În această aplicație, releele sunt utilizate ca actuatori pentru a controla alimentarea dispozitivelor pe baza semnalelor de la microcontroler. Releele permit izolarea circuitului de comandă de cel de putere și asigură un control sigur și fiabil al echipamentelor conectate. Actuatorii reprezintă o componentă critică în atingerea și menținerea stabilității sistemului, prevenind oscilarea rapidă a valorii controlate și protejând echipamentele de o uzură excesivă datorită ciclurilor de pornire/oprire repetate.

EFECTUAREA LUCRĂRII

Materiale necesare:

- **Microcontroler** (Arduino Mega) folosit ca microcontroler principal pentru controlul releului și afișarea stării pe LCD. Acesta va primi comenzi prin interfața serială și va controla becul;
- **Releu** va fi utilizat pentru a comanda aprinderea și stingerea LED-ului, care simulează becul electric în această configurație;
- **LED** folosit pentru a reprezenta vizual becul aprins/stins. Acesta va fi conectat la terminalul NO (normally open) al releului, ceea ce permite LED-ului să se aprindă doar când releul este activat;
- **Display LCD I2C** pentru a afișa starea becului (aprins sau stins). Interfața I2C simplifică conexiunile necesare pentru LCD, reducând numărul de pini necesari pe Arduino Mega;
- **Surse de alimentare și fire de conectare** asigură alimentarea circuitului și conectarea între componente;
- **Modul de alimentare (VCC)** – în acest caz, simbolul de alimentare „VCC” este un nod virtual ce asigură o conexiune constantă de 5V;
- **Senzor NTC -** sensor analog de temperatură (Negative Temperature Coefficient) thermistor;
- **2 Butoane** – pentru controlul SetPoint-ului.

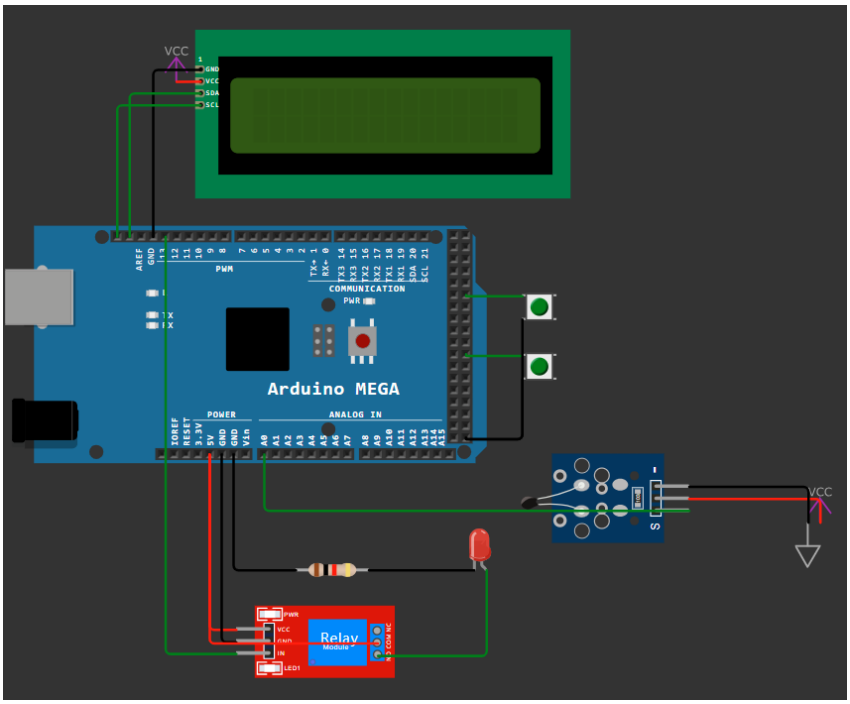

Figura 1 – Ansamblarea circuitului virtual

Modul de lucru

```
void setup() {
  setup_button(INC_BUTTON_PIN);
  setup_button(DEC_BUTTON_PIN);
  setup_relay(RELAY_PIN);
  initializeLcd();
  initializeSerial(115200);

  lcdSemaphore = xSemaphoreCreateBinary();
  serialSemaphore = xSemaphoreCreateBinary();

  xTaskCreate(Task_to_SetPoint, "Set the desired temperature", 1000, NULL, 1, NULL);
  xTaskCreate(Task_to_ControlRelay, "Control relay on or off", 1000, NULL, 1, NULL);
  xTaskCreate(Task_to_ReadTemperature, "Read room temperature in Celsius", 1000, NULL,
1, NULL);
  xTaskCreate(Task_to_PrintOnLcd, "Idle task to print values on LCD", 1000, NULL, 1,
NULL);
  xTaskCreate(Task_to_SendtoPlotter, "Idle task to print values on Serial", 1000, NULL,
1, NULL);
}
```

În setup(), inițializăm butoanele, releul, LCD-ul, și serial-ul, și creăm semafoarele necesare pentru sincronizarea task-urilor. De asemenea, sunt create sarcinile (task-urile) FreeRTOS care se vor ocupa de ajustarea setpoint-ului, controlul releului, citirea temperaturii, afișarea pe LCD, și trimiterea datelor la Serial.

Funcțiile de Task, servesc drept sarcini ce vor rula în paralel folosind FreeRTOS:

```
void Task_to_SetPoint();
void Task_to_ControlRelay();
void Task_to_ReadTemperature();
void Task_to_PrintOnLcd();
void Task_to_SendtoPlotter();
```

Acestea sunt însoțite de funcțiile helper care descriu procesele ca:

```
void SetPoint() {
  if (isPressed = buttonPressed(INC_BUTTON_PIN)) {
    if (setPoint < MAX_TEMPERATURE) {
      setPoint += step;
    }
  }
  else if (isPressed = buttonPressed(DEC_BUTTON_PIN)) {
    if (setPoint > MIN_TEMPERATURE) {
      setPoint -= step;
    }
  }
}
```

Funcție care detectează apăsarea butoanelor pentru creșterea sau scăderea setpoint-ului, cu verificarea limitelor maxime și minime.

```
void ControlRelay() {
  if (setPoint >= temperature + HYSTERESIS) {
    relayState = turnRelayOff(RELAY_PIN);
  }
  else if (setPoint <= temperature - HYSTERESIS) {
    relayState = turnRelayOn(RELAY_PIN);
  }
}
```

Funcția ControlRelay controlează starea releului pe baza diferenței dintre temperatura curentă și setpoint, luând în considerare histerezisul.

```
void ReadTemperature() {
  temperature = get_temperature_directly();
}
```

Această funcție citește temperatura de la senzorul NTC.

```
void PrintOnLcd() {
  String message1 = "Curent: " + String(temperature, 2);
  String message2 = "SetPoint: " + String(setPoint, 2);

  resetLcd(message1.c_str(), message2.c_str());
}
```

Funcția PrintOnLcd afișează pe LCD valorile curente ale temperaturii și setpoint-ului.

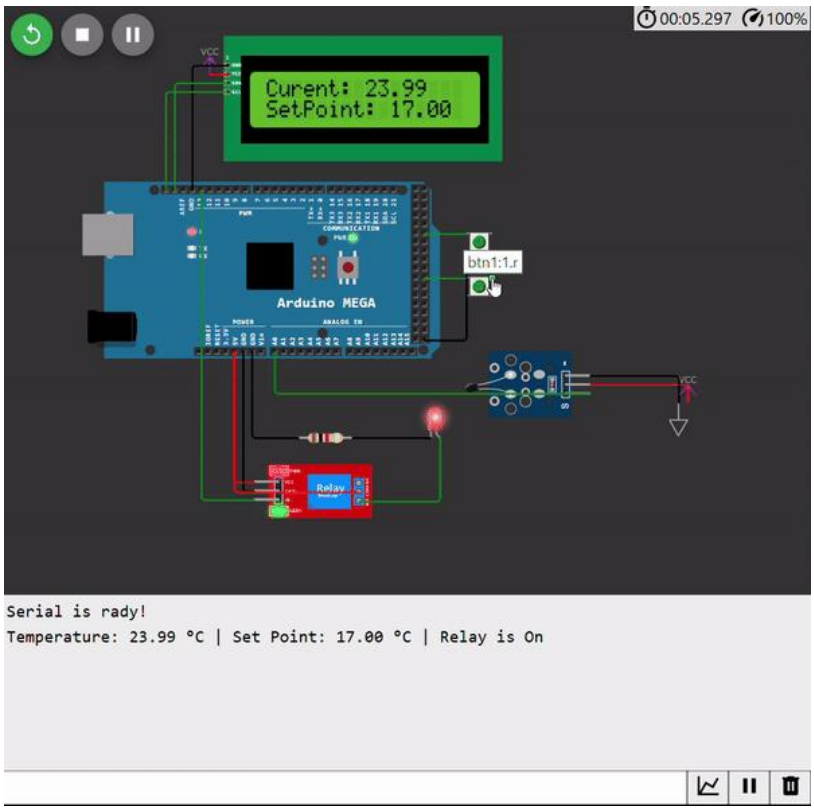

Figura 2 – Simularea aplicației în wokwi

CONCLUZIE

Lucrarea de laborator a demonstrat eficiența controlului funcțional de tip ON-OFF cu histerezis într-un sistem de monitorizare a temperaturii, utilizând un microcontroler Arduino Mega, senzori și un releu. Am implementat cu succes un sistem de control bazat pe setarea unei temperaturi de referință și pe activarea/dezactivarea releului în funcție de abaterile temperaturii curente față de această valoare, ținând cont de histerezis pentru a reduce numărul de cicluri de comutare și a proteja componentele. De asemenea, am folosit un ecran LCD I2C pentru afișarea valorilor curente și de setpoint, iar butoanele de creștere și scădere au permis ajustarea valorii de referință.

Utilizarea sistemului de operare FreeRTOS a facilitat crearea de sarcini (task-uri) independente pentru citirea temperaturii, actualizarea valorilor pe ecran, și trimiterea datelor prin interfața serială, asigurând astfel un control stabil și flexibil. Această abordare modulară contribuie la scalabilitatea sistemului și permite extinderea ușoară a funcționalității, cum ar fi adăugarea de noi senzori sau ajustarea parametrilor de control.

În concluzie, acest proiect ilustrează avantajele controlului ON-OFF cu histerezis într-un sistem de monitorizare a temperaturii și ar putea servi drept bază pentru dezvoltarea unor aplicații similare în domeniul industrial sau casnic, acolo unde este necesară menținerea unor condiții ambientale constante și economice.

BIBLIOGRAFII

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;
4. Proiectul pe GitHub, Resursa electronică: https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator5 - Regim de acces;

Anexa 1

Fișierul stdinout.h

```
#ifndef _STDINOUT_H
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
        static size_t initnum;
public:
        // Constructor
        initializeSTDINOUT();
};

// Call the constructor in each compiled file this header is included in
// static means the names won't collide
static initializeSTDINOUT initializeSTDINOUT_obj;

#endif
```

Fișierul stdinout.cpp

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <stdio.h>
#include "stdinout.h"

// Function that printf and related will use to print
static int serial_putchar(char c, FILE *f)
{
  if (c == '\n') {
    serial_putchar('\r', f);
  }

  return Serial.write(c) == 1 ? 0 : 1;
}
// Function that scanf and related will use to read
static int serial_getchar(FILE *)
{
  // Wait until character is avilable
  while (Serial.available() <= 0) { ; }

  return Serial.read();
}

static FILE serial_stdinout;

static void setup_stdin_stdout()
```

```
{
  // Set up stdout and stdin
  fdev_setup_stream(&serial_stdinout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
  stdout = &serial_stdinout;
  stdin  = &serial_stdinout;
  stderr = &serial_stdinout;
}

// Initialize the static variable to 0
size_t initializeSTDINOUT::initnum = 0;

// Constructor that calls the function to set up stdin and stdout
initializeSTDINOUT::initializeSTDINOUT()
{
  if (initnum++ == 0) {
    setup_stdin_stdout();
  }
}
```

Anexa 2

main.cpp

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include "lcd_utils.h"
#include "serial_utils.h"
#include "relay.h"
#include "button_utils.h"
#include "NTC.h"

#define RELAY_PIN 13
#define INC_BUTTON_PIN 31
#define DEC_BUTTON_PIN 41
#define HYSTERESIS 2.0
#define MAX_TEMPERATURE 80
#define MIN_TEMPERATURE -24

bool isPressed = false;
double temperature = 0;
double lastTempValue = temperature;
double setPoint = 17.0;
double step = 1;
bool relayState = false;
bool lastRelayState = relayState;

void Task_to_SetPoint();
void Task_to_ControlRelay();
void Task_to_ReadTemperature();
void Task_to_PrintOnLcd();
void Task_to_SendtoPlotter();

SemaphoreHandle_t lcdSemaphore;
SemaphoreHandle_t serialSemaphore;
```

```
void setup() {
  setup_button(INC_BUTTON_PIN);
  setup_button(DEC_BUTTON_PIN);
  setup_relay(RELAY_PIN);
  initializeLcd();
  initializeSerial(115200);

  lcdSemaphore = xSemaphoreCreateBinary();
  serialSemaphore = xSemaphoreCreateBinary();

  xTaskCreate(Task_to_SetPoint, "Set the desired temperature", 1000, NULL, 1, NULL);
  xTaskCreate(Task_to_ControlRelay, "Control relay on or off", 1000, NULL, 1, NULL);
  xTaskCreate(Task_to_ReadTemperature, "Read room temperature in Celsius", 1000, NULL,
1, NULL);
  xTaskCreate(Task_to_PrintOnLcd, "Idle task to print values on LCD", 1000, NULL, 1,
NULL);
  xTaskCreate(Task_to_SendtoPlotter, "Idle task to print values on Serial", 1000, NULL,
1, NULL);
}

void loop() {
  // SetPoint();
  // ControlRelay();
  // ReadTemperature();
  // PrintOnLcd();
  // delay(2000);
  // Serial.println(isPressed);
  // isPressed = false;
}

void Task_to_SetPoint() {
  while (true) {
    SetPoint();
    if (isPressed) {
      isPressed = false;
      xSemaphoreGive(lcdSemaphore);
    }
    vTaskDelay(200 / portTICK_PERIOD_MS );
  }
}

void Task_to_ControlRelay() {
  while (true) {
    ControlRelay();
    if(lastRelayState != relayState){
      xSemaphoreGive(serialSemaphore);
      lastRelayState = relayState;
    }
    vTaskDelay(500 / portTICK_PERIOD_MS );
  }
}
```

```
void Task_to_ReadTemperature() {
  while (true) {
    ReadTemperature();
    if (temperature != lastTempValue) {
      lastTempValue = temperature;
      xSemaphoreGive(lcdSemaphore);
    }
    vTaskDelay(200 / portTICK_PERIOD_MS );
  }
}
void Task_to_PrintOnLcd() {
  while (1) {
    if (xSemaphoreTake(lcdSemaphore, portMAX_DELAY) == pdTRUE) {
      PrintOnLcd();
    }
  }
}

void Task_to_SendtoPlotter() {
  char buffer[10];
  while (1) {
    if (xSemaphoreTake(serialSemaphore, portMAX_DELAY) == pdTRUE) {
      dtostrf(temperature, 5, 2, buffer);
      printf("Temperature: %s °C | ", buffer);
      dtostrf(setPoint, 5, 2, buffer);
      printf("Set Point: %s °C | ", buffer);
      printf("Relay is %s", relayState == HIGH ? "On" : "Off");
      printf("\n");
    }
  }
}

void SetPoint() {
  if (isPressed = buttonPressed(INC_BUTTON_PIN)) {
    if (setPoint < MAX_TEMPERATURE) {
      setPoint += step;
    }
  }
  else if (isPressed = buttonPressed(DEC_BUTTON_PIN)) {
    if (setPoint > MIN_TEMPERATURE) {
      setPoint -= step;
    }
  }
}

void ControlRelay() {
  if (setPoint >= temperature + HYSTERESIS) {
    relayState = turnRelayOff(RELAY_PIN);
  }
  else if (setPoint <= temperature - HYSTERESIS) {
    relayState = turnRelayOn(RELAY_PIN);
  }
```

```
}

void ReadTemperature() {
  temperature = get_temperature_directly();
}

void PrintOnLcd() {
  String message1 = "Curent: " + String(temperature, 2);
  String message2 = "SetPoint: " + String(setPoint, 2);

  resetLcd(message1.c_str(), message2.c_str());
}
```

relay.h

```
#ifndef RELAY_H
#define RELAY_H

#include <Arduino.h>

bool turnRelayOn(short pin);
bool turnRelayOff(short pin);
void setup_relay(short pin);

#endif
```

relay.cpp

```
#include "relay.h"

void setup_relay(short pin){
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
}

bool turnRelayOn(short pin) {
  digitalWrite(pin, HIGH);
  //printf("Relay ON\n");
  return true;
}

bool turnRelayOff(short pin) {
  digitalWrite(pin, LOW);
  //printf("Relay OFF\n");
  return false;
}
```

lcd_utils.h

```
#ifndef LCD_UTILS_H
#define LCD_UTILS_H

#include <LiquidCrystal_I2C.h>

#define I2C_ADDR    0x27
#define COLUMNS 16
#define ROWS   2

void initializeLcd();
void resetLcd(const char* temp, const char* setPoint);

#endif // LCD_UTILS_H
```

lcd_utils.cpp

```
#include "lcd_utils.h"
#include <stdio.h>
LiquidCrystal_I2C lcd(I2C_ADDR, COLUMNS, ROWS);

void initializeLcd(){
  lcd.begin(16, 2);
  lcd.backlight();
  resetLcd("", "");
}

void resetLcd(const char* temp, const char* setPoint) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(temp);
  lcd.setCursor(0, 1);
  lcd.print(setPoint);
}
```

serial_utils.h

```
#ifndef SERIAL_UTILS_H
#define SERIAL_UTILS_H

#include <Arduino.h>

void initializeSerial(int64_t baud);
int16_t getNumericValue();

#endif // SERIAL_UTILS_H
```

serial_utils.cpp

```
#include "serial_utils.h"

void initializeSerial(int64_t baud){
  Serial.begin(baud);
  printf("Serial is rady!\n");
```

```
}

int16_t getNumericValue() {
  char input[10];  // Șir pentru a stoca inputul utilizatorului
  int16_t speed = 0;
  bool valid = false;

  while (!valid) {
    // Citim inputul ca un șir de caractere
    scanf("%s", input);

    // Verificăm dacă inputul este numeric și poate începe cu '-' sau '+'
    valid = true;  // Presupunem că este valid până la dovada contrară
    int i = 0;
    if (input[0] == '-' || input[0] == '+') {
      i = 1;  // Dacă începe cu '-' sau '+', începem verificarea de la următorul caracter
    }

    // Verificăm restul caracterelor să fie doar cifre
    for (; input[i] != '\0'; i++) {
      if (input[i] < '0' || input[i] > '9') {
        valid = false;
        break;
      }
    }

    // Dacă inputul este valid, convertim la număr și verificăm intervalul
    if (valid) {
      speed = atoi(input);  // Convertim șirul la număr întreg

    } else {
      printf("Invalid input. Please enter a numeric value.\n");
    }
  }
  return speed;  // Returnăm valoarea introdusă
}
```

NTC.h

```
#ifndef NTC_H
#define NTC_H

#include <Arduino.h>

#define BETA 3950 // valoarea BETA a termistorului NTC
#define ANALOG_PIN A0 //pinul analogic de conexiune
#define V_REF 5.0 // tensiunea de referință utilizată de ADC
#define R_REF 10000.0 // rezistența de referință utilizată în divizorul de tensiune
#define ADC_MAX 1023.0 // valoarea maximă pe care o poate avea un ADC de 10 biți

double get_temperature_directly();
double adc_to_Voltage(int16_t adcValue);
```

```
double calculate_ntc_resistance(double voltage);
double convert_to_Kelvin(double ntcResistance);
double convert_to_Celsius(double ntcResistance);

#endif //NTC_h
```

NTC.cpp

```
#include "NTC.h"

double get_temperature_directly() {
  int16_t adcValue = analogRead(ANALOG_PIN);
  double temperature = 1 / (log(1 / (ADC_MAX / adcValue - 1))
              / BETA + 1.0 / 298.15) - 273.15;
  return temperature;
}

double adc_to_Voltage(int16_t adcValue) {
  return (adcValue / ADC_MAX) * V_REF;
}

double calculate_ntc_resistance(double voltage) {
  return R_REF * (V_REF / voltage - 1);
}

double convert_to_Kelvin(double ntcResistance) {
  return 1.0 / (log( R_REF / ntcResistance  ) / BETA + 1.0 / 298.15);
}

double convert_to_Celsius(double ntcResistance) {
  return convert_to_Kelvin(ntcResistance) - 273.15;
}
```

button_utils.h

```
#ifndef BUTTON_UTILS_H
#define BUTTON_UTILS_H

#include <Arduino.h>

void setup_button(uint8_t pin);
bool buttonPressed(uint8_t pin);

#endif
```

button_utils.cpp

```
#include "lcd_utils.h"
#include <stdio.h>
LiquidCrystal_I2C lcd(I2C_ADDR, COLUMNS, ROWS);

void initializeLcd(){
  lcd.begin(16, 2);
```

```
  lcd.backlight();
  resetLcd("", "");
}

void resetLcd(const char* temp, const char* setPoint) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(temp);
  lcd.setCursor(0, 1);
  lcd.print(setPoint);
}
```

ANEXA 3

diagram.json

```
{
  "version": 1,
  "author": "Mihai Muntean",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-mega", "id": "mega", "top": 115.8, "left": -224.4, "attrs":
{} },
    {
      "type": "wokwi-led",
      "id": "led1",
      "top": 351.6,
      "left": 138.2,
      "attrs": { "color": "red" }
    },
    {
      "type": "wokwi-pushbutton-6mm",
      "id": "btn1",
      "top": 218.6,
      "left": 192,
      "attrs": { "color": "green" }
    },
    {
      "type": "wokwi-pushbutton-6mm",
      "id": "btn2",
      "top": 170.6,
      "left": 192,
      "attrs": { "color": "green" }
    },
    {
      "type": "wokwi-resistor",
      "id": "r1",
      "top": 387.95,
      "left": 9.6,
      "attrs": { "value": "1000" }
    },
    {
```

```
      "type": "wokwi-lcd1602",
      "id": "lcd1",
      "top": -41.6,
      "left": -71.2,
      "attrs": { "pins": "i2c" }
    },
    { "type": "wokwi-relay-module", "id": "relay1", "top": 422.6, "left": -38.4, "attrs":
{} },
    { "type": "wokwi-vcc", "id": "vcc1", "top": -28.04, "left": -96, "attrs": {} },
    {
      "type": "wokwi-ntc-temperature-sensor",
      "id": "ntc1",
      "top": 300.2,
      "left": 191.4,
      "attrs": {}
    },
    { "type": "wokwi-vcc", "id": "vcc2", "top": 327.16, "left": 422.4, "attrs": {} },
    { "type": "wokwi-gnd", "id": "gnd1", "top": 355.2, "left": 412.2, "attrs": {} }
  ],
  "connections": [
    [ "relay1:NO", "led1:A", "green", [ "h0" ] ],
    [ "lcd1:GND", "mega:GND.1", "black", [ "h0" ] ],
    [ "vcc1:VCC", "lcd1:VCC", "red", [ "v0" ] ],
    [ "lcd1:SDA", "mega:SDA", "green", [ "h0" ] ],
    [ "lcd1:SCL", "mega:SCL", "green", [ "h0" ] ],
    [ "relay1:VCC", "mega:5V", "red", [ "h0" ] ],
    [ "relay1:GND", "mega:GND.2", "black", [ "h0" ] ],
    [ "relay1:IN", "mega:13", "green", [ "h0" ] ],
    [ "mega:5V", "relay1:COM", "red", [ "v0" ] ],
    [ "mega:GND.3", "r1:1", "black", [ "v0" ] ],
    [ "r1:2", "led1:C", "black", [ "v0" ] ],
    [ "btn2:1.l", "mega:31", "green", [ "h0" ] ],
    [ "btn1:1.l", "mega:41", "green", [ "h0" ] ],
    [ "mega:GND.5", "btn1:2.l", "black", [ "v0.95", "h45.4" ] ],
    [ "mega:GND.5", "btn2:2.l", "black", [ "v0.95", "h45.4" ] ],
    [ "ntc1:VCC", "vcc2:VCC", "red", [ "h0" ] ],
    [ "gnd1:GND", "ntc1:GND", "black", [ "v0" ] ],
    [ "mega:A0", "ntc1:OUT", "green", [ "v0" ] ]
  ],
  "dependencies": {}
}
```