**1. Розробка алгоритму ро­боти та програми для контролера акумулятора**

Розробка алгоритму роботи прошивки тюнера є важливим кроком для розуміння принципів роботи пристрою, який розробляється. Блок схема алгоритму зображена на рисунку 1.

Алгоритм роботи прошивки контролер акумулятора

1. Ініціалізація усіх змінних та констант які будуть потрібні для роботи програми.
2. Ініціалізація дисплею.
3. Підключення необхідних бібліотек.
4. Ініціалізація пінів Arduino.
5. Активація внутрішніх переривань мікроконтролера для формування потрібних затримок і використання таймера-лічильника.
6. Розробка логіки для датчика температури.
7. Розробка логіки для основної роботи контролера акумулятора: розрахунок та індикація заряду, попередження про низький заряд.
8. Розробка логіки для детектування газу і оповіщення про це.

Програма для забезпечення роботи контролера акумулятора згідно алгоритму наведеного на рисунку 2 наведена нижче. Дана програма може коригуватися в залежності від використання пристрою для якого буде застосовуватися контролер акумулятора.

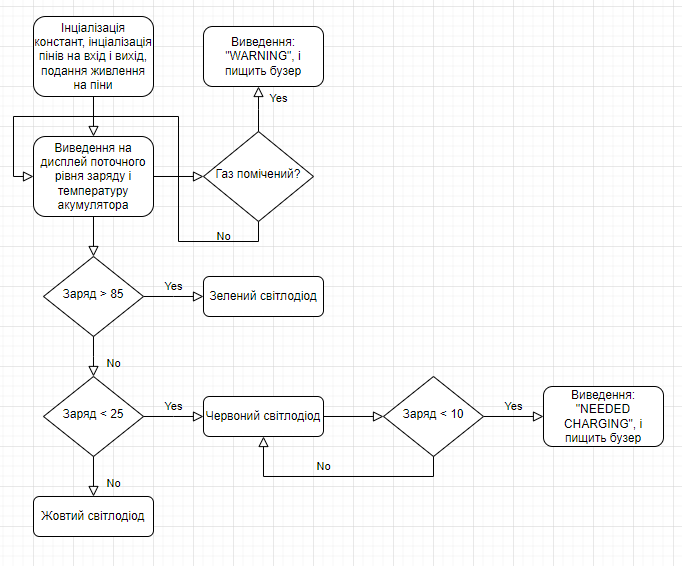


Рисунок 1 – Блок схема алгоритму роботи контролера акумулятора

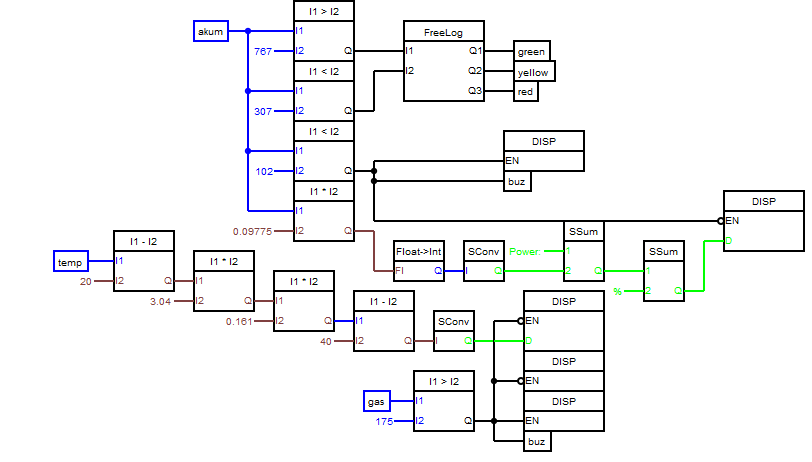


Рисунок 2 – Скрін з середовища розробки FL Prog

**Код програми:**

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C \_lcd1(0x27, 16, 2);

int \_dispTempLength1=0;

boolean \_isNeedClearDisp1;

bool \_FreeLog2\_IArr[2];

bool \_FreeLog2\_Q1\_StArr[] = {1, 0};

bool \_FreeLog2\_Q2\_StArr[] = {0, 0};

bool \_FreeLog2\_Q3\_StArr[] = {0, 1};

int \_disp5oldLength = 0;

int \_disp2oldLength = 0;

bool \_FreeLog2\_Q1 = 0;

bool \_FreeLog2\_Q2 = 0;

bool \_FreeLog2\_Q3 = 0;

int \_disp1oldLength = 0;

int \_disp3oldLength = 0;

int \_disp4oldLength = 0;

void setup()

{

pinMode(2, OUTPUT);

digitalWrite(2, 0);

pinMode(3, OUTPUT);

digitalWrite(3, 0);

pinMode(4, OUTPUT);

digitalWrite(4, 0);

pinMode(9, OUTPUT);

digitalWrite(9, 0);

Wire.begin();

delay(10);

\_lcd1.init();

\_lcd1.backlight();

}

void loop()

{

if (\_isNeedClearDisp1)

{

\_lcd1.clear();

\_isNeedClearDisp1= 0;

}

//Плата:1

\_FreeLog2\_IArr[0] = ((analogRead (2))) > (767);

\_FreeLog2\_IArr[1] = ((analogRead (2))) < (307);

\_FreeLog2\_Q1 = \_checkFreeLogicBlockOutput(\_FreeLog2\_IArr, 2, \_FreeLog2\_Q1\_StArr, 2);

\_FreeLog2\_Q2 = \_checkFreeLogicBlockOutput(\_FreeLog2\_IArr, 2, \_FreeLog2\_Q2\_StArr, 2);

\_FreeLog2\_Q3 = \_checkFreeLogicBlockOutput(\_FreeLog2\_IArr, 2, \_FreeLog2\_Q3\_StArr, 2);

digitalWrite(4, \_FreeLog2\_Q3);

digitalWrite(3, \_FreeLog2\_Q2);

digitalWrite(2, \_FreeLog2\_Q1);

digitalWrite(9, ((analogRead (2))) < (102));

if (((analogRead (2))) < (102))

{

\_dispTempLength1 = (String("NEEDED CHARGING")).length();

if (\_disp1oldLength > \_dispTempLength1)

{

\_isNeedClearDisp1 = 1;

}

\_disp1oldLength = \_dispTempLength1;

\_lcd1.setCursor(int((16 - \_dispTempLength1)/2), 0);

\_lcd1.print(String("NEEDED CHARGING"));

}

else

{

if (\_disp1oldLength > 0)

{

\_isNeedClearDisp1 = 1;

\_disp1oldLength = 0;

}

}

if (!(((analogRead (2))) < (102)))

{

\_dispTempLength1 = ((((((String("Power: ")) + ((String((int(((analogRead (2)))\*(0.09775))), DEC))))) + (String(" %"))))).length();

if (\_disp5oldLength > \_dispTempLength1)

{

\_isNeedClearDisp1 = 1;

}

\_disp5oldLength = \_dispTempLength1;

\_lcd1.setCursor(int((16 - \_dispTempLength1)/2), 0);

\_lcd1.print((((((String("Power: ")) + ((String((int(((analogRead (2)))\*(0.09775))), DEC))))) + (String(" %")))));

}

else

{

if (\_disp5oldLength > 0)

{

\_isNeedClearDisp1 = 1;

\_disp5oldLength = 0;

}

}

digitalWrite(9, ((analogRead (0))) > (175));

if (((analogRead (0))) > (175))

{

\_dispTempLength1 = (String("WARNING")).length();

if (\_disp4oldLength > \_dispTempLength1)

{

\_isNeedClearDisp1 = 1;

}

\_disp4oldLength = \_dispTempLength1;

\_lcd1.setCursor(int((16 - \_dispTempLength1)/2), 1);

\_lcd1.print(String("WARNING"));

}

else

{

if (\_disp4oldLength > 0)

{

\_isNeedClearDisp1 = 1;

\_disp4oldLength = 0;

}

}

if (!(((analogRead (0))) > (175)))

{

\_dispTempLength1 = (String("Temp.= C")).length();

if (\_disp3oldLength > \_dispTempLength1)

{

\_isNeedClearDisp1 = 1;

}

\_disp3oldLength = \_dispTempLength1;

\_lcd1.setCursor(0, 1);

\_lcd1.print(String("Temp.= C"));

}

else

{

if (\_disp3oldLength > 0)

{

\_isNeedClearDisp1 = 1;

\_disp3oldLength = 0;

}

}

if (!(((analogRead (0))) > (175)))

{

\_dispTempLength1 = (((\_floatToStringWitRaz((((((analogRead (1)))-(20.00))\*(3.04))\*(0.161))-(40.00),2)))).length();

if (\_disp2oldLength > \_dispTempLength1)

{

\_isNeedClearDisp1 = 1;

}

\_disp2oldLength = \_dispTempLength1;

\_lcd1.setCursor(7, 1);

\_lcd1.print(((\_floatToStringWitRaz((((((analogRead (1)))-(20.00))\*(3.04))\*(0.161))-(40.00),2))));

}

else

{

if (\_disp2oldLength > 0)

{

\_isNeedClearDisp1 = 1;

\_disp2oldLength = 0;

}

}

}

String \_floatToStringWitRaz(float value, int raz)

{

return String(value,raz);

}

bool \_checkFreeLogicBlockOutput(bool inArray[], int inArraySize, bool stArray[], int stArraySize)

{

int inIndex = 0;

bool result = 1;

for (int i = 0; i < stArraySize; i = i + 1)

{

if (!(inArray[inIndex] == stArray[i]))

{

result = 0;

}

inIndex ++;

if (inIndex == inArraySize)

{

if (result)

{

return 1;

}

else

{

result = 1;

}

inIndex = 0;

}

}

return 0;

}

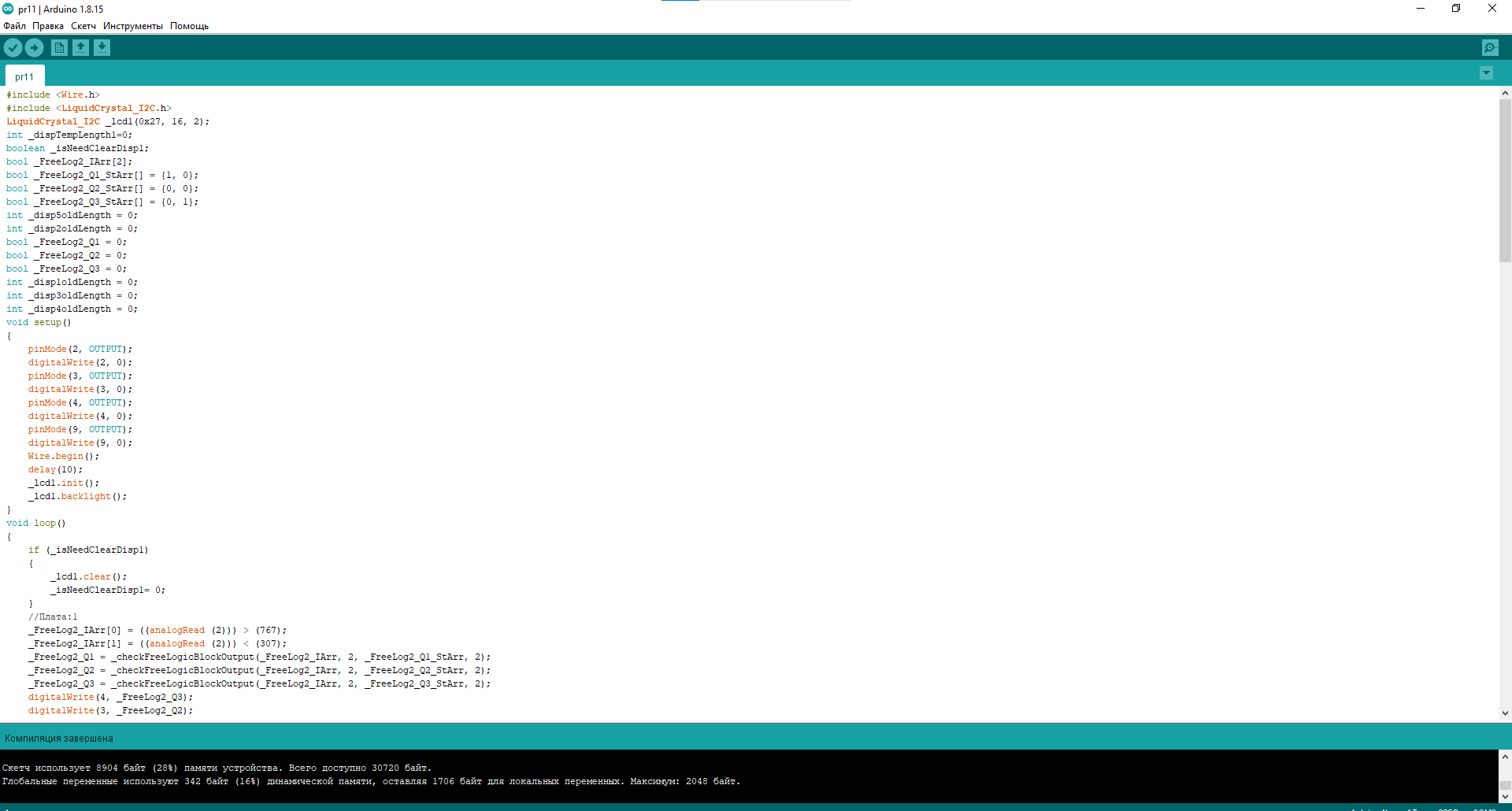


Рисунок 3 – Скрін успішної компіляції прошивки мікроконтролера

**2. Складання схеми у симуляторі та перевірка роботи контролера акумулятора**

Симуляція роботи і візуальне зображення контролера акумулятора буде виконуватися в Tinkercad. Для сумісності між реальним та віртуальним пристроєм використаємо такі ж самі елементи, які повинні бути б використані для розробки реального пристрою, з’єднавши все згідно схеми електричної принципової. Досліджувана схема зображена на рисунку 4.

Порядок симуляції роботи схеми:

* зібрати схему в Tinkercad;
* закачати прошивку до мікроконтролера Arduino NANO;
* перевірка режимів роботи досліджуваної схеми;

Режими роботи поділено на:

* Запуск
* Сигналізацію рівня заряду світлодіодами і бузером
* Робота термометра
* Попередження про наявний газ

Моделювання в Tinkercad дає можливість показати усі режими роботи пристрою. Але недоліком такого моделювання є, що не факт що на реальному «залізі» воно буде коректно працювати.

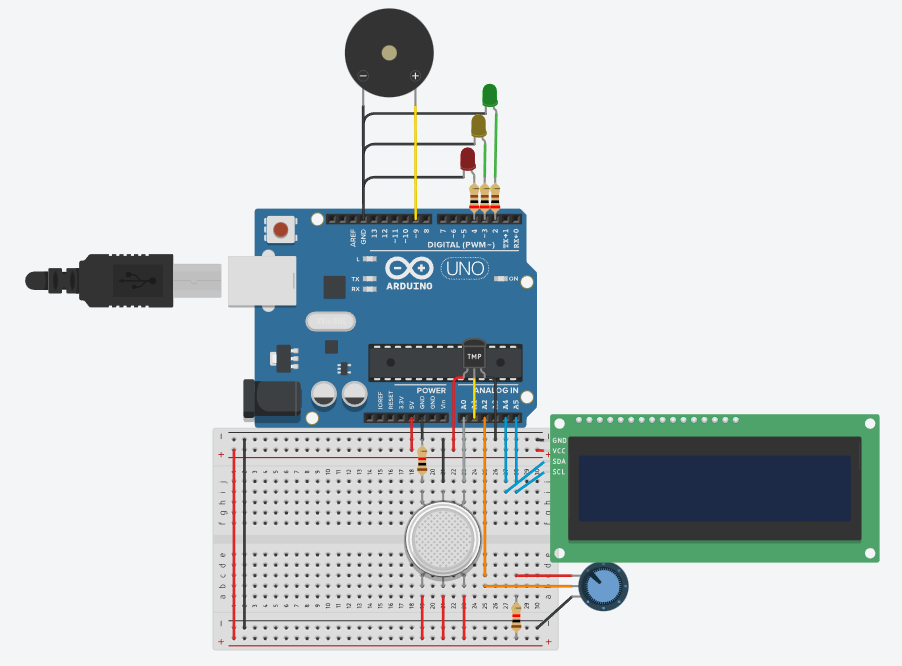


Рисунок 4 – Схема контролера акумулятора

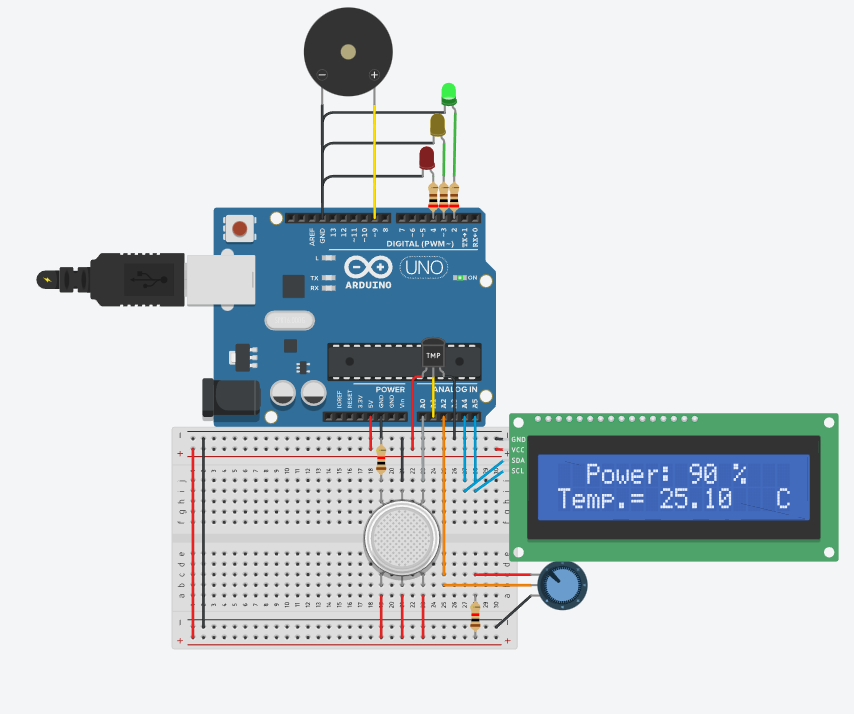


Рисунок 5 – Запуск приладу

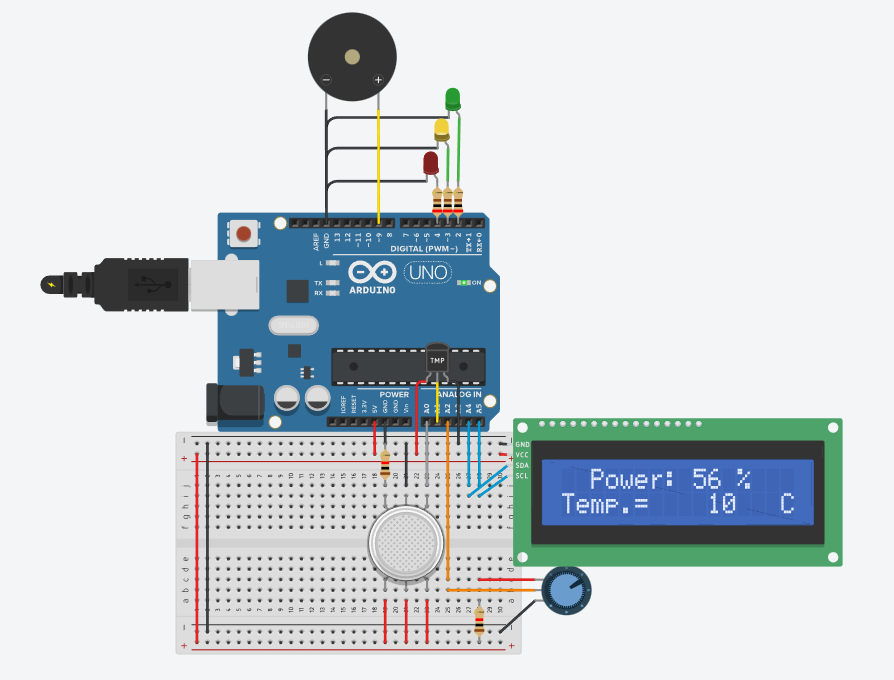


Рисунок 6 – Рівень заряду менше 60

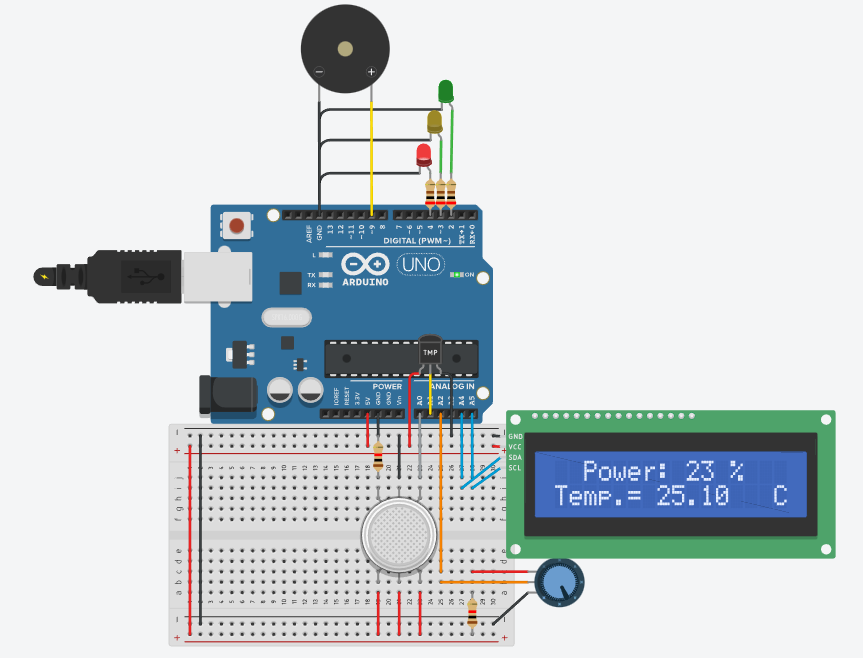


Рисунок 7 – Рівень заряду менше 30

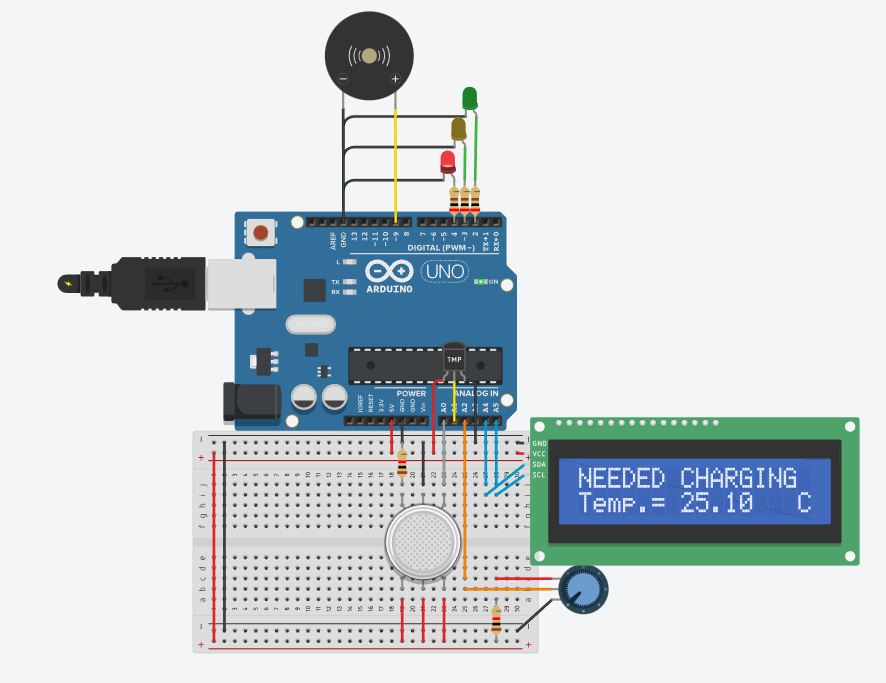


Рисунок 8 – Рівень заряду менше 10

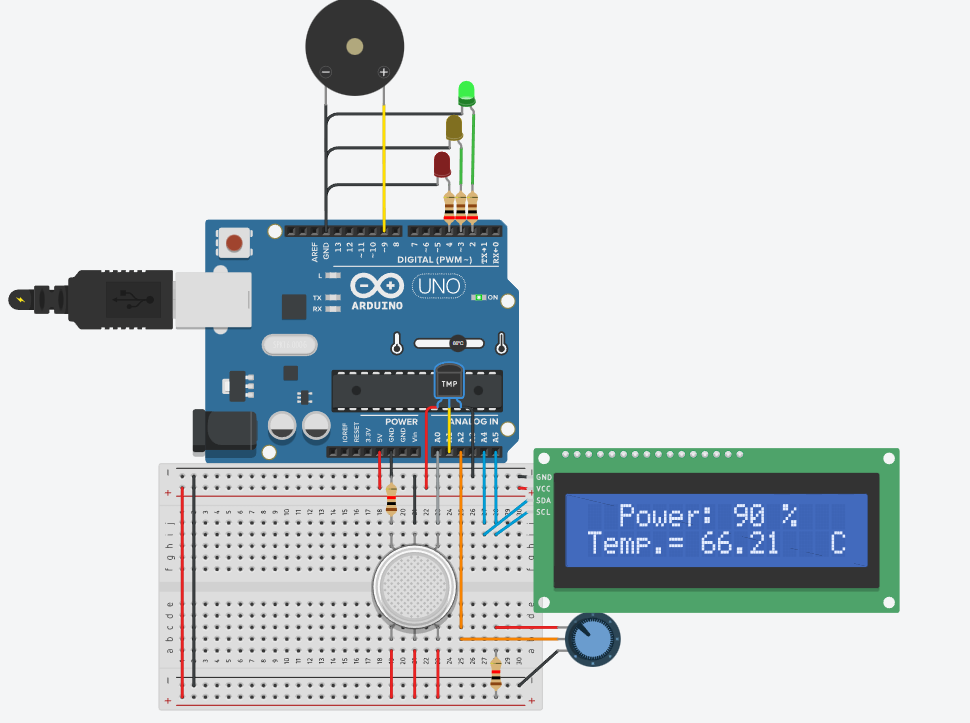


Рисунок 9 – Зміна температури

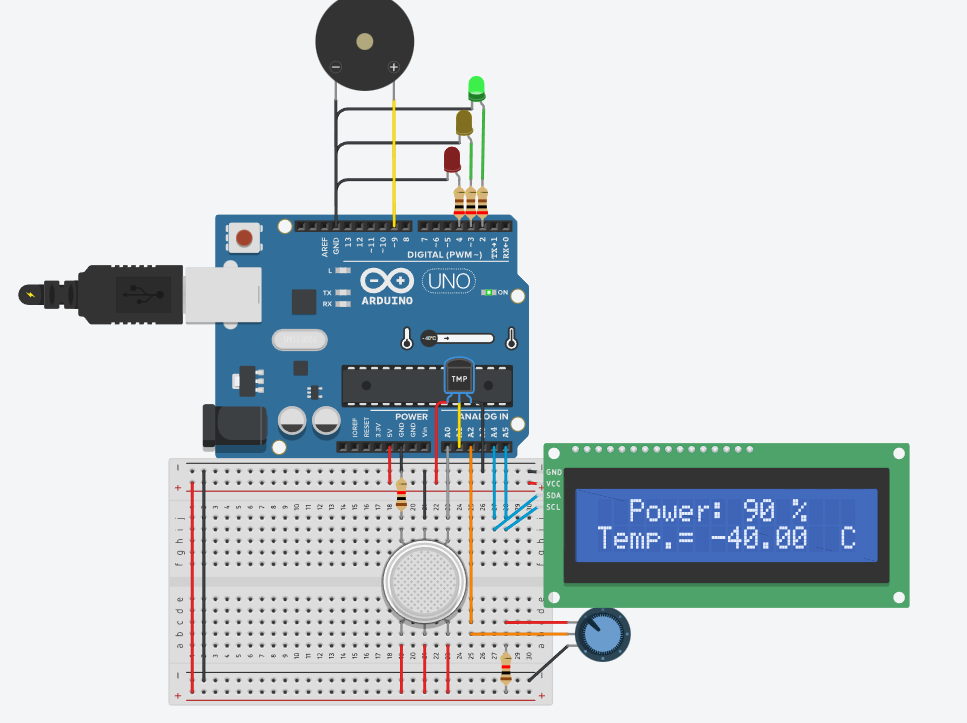


Рисунок 10– Зміна температури

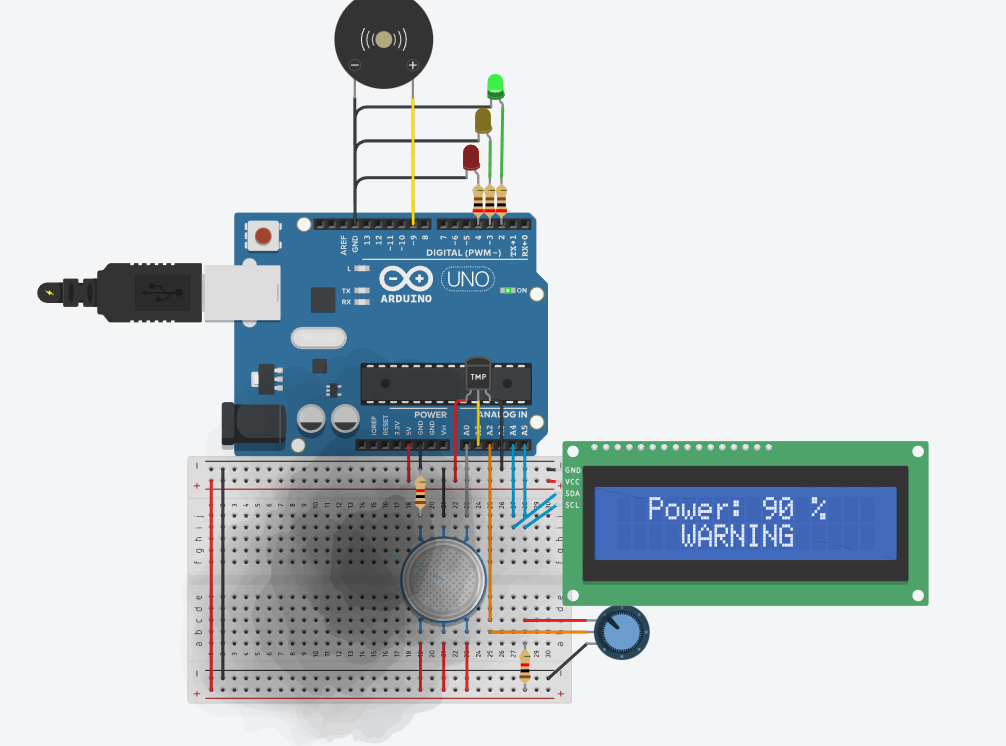


Рисунок 11 – Наявність газу

Рекомендації щодо використання приладу.

Прилад можна використовувати можна використовувати, для контролю автомобільного акумулятора, вимірювати його температуру та заряд і вивід на передню панель автомобіля. Доволі корисний прилад для розширення функціоналу передньої панелі і догляду за акумулятором. Також в разі виникнення загоряння чогось під капотом буде виділятись дим, який «замітить» датчик газу і попередить власника автомобіля.

**ВИСНОВОК**

В результаті проходження практики був розроблений контролер акумулятора, на основі мікроконтролера Arduino Nano, яка працює таким чином: при запуску виводить на дисплей поточний заряд і температуру акумулятора, якщо заряд перетинає деякий поріг змінюється колір світлодіода: зелений світлодіод має світитися, коли рівень заряду більше 75%, жовтий світлодіод, коли рівень заряду більше 30% і червоний світлодіод в інших видках. Коли рівень заряду буде менше 10% має спрацьовувати звукова сигналізація із виведенням інформації на LCD екран про необхідність зарядки. Також при появі газу (наприклад диму), давач газу його фіксує і оповіщає через бузер і повідомленням на дисплей LCD.

В першому розділі була розроблена структурна схема, яка складається з таких блоків: джерело живлення, Arduino NANO, 3-ох світлодіодів (червоного, жовтого та зеленого), бузера, давача температури, давача газу, потенціометра, I2C конвертора та LCD дисплею. Основною частиною приладу є Arduino NANO, що відповідає безпосередньо за керування всією системою.

В другому розділі розроблено алгоритм роботи програми та код для програмування Arduino Nano для пристрою.

Практичне виконання пристрою, де для початку було виконане моделювання в Tinkercad. Потім розроблено схему на макетній платі та тестування готового пристрою.