МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ «ФАХОВИЙ КОЛЕДЖ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

КУРСОВИЙ ПРОЄКТ

на тему:

«Виготовлення охоронного пристрою на платі Arduino з використанням датчиків HC-SR501 І HC-SR04 та виводом інформації на LCD дисплей»

Виконав студент групи ІП-41
Максим ЛЕГАН
Керівник проєкту:
Остап ЮНАК
Курсовий проєкт перевірений
і допущений до захисту
“___” _____2025 р.
Курсовий проєкт при захисті оцінений
_____

Львів 2025

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ЗАТВЕРДЖЕНО
на засіданні циклової комісії
«Фундаментальної підготовки»
Протокол № __ від _____ 2025 р.
Голова комісії
_____ Богдан ПЕЛЕЩАК

ЗАВДАННЯ

на курсовий проєкт

Максиму Легану Михайловичу
(прізвище, ім'я та по батькові)

з навчальної дисципліни: ПРОГРАМНО-АПАРАТНІ ЗАСОБИ INTERNET-РЕЧЕЙ
Студент групи: ІП-41
1. Тема проєкту: Виготовлення охоронного пристрою на платі Arduino з використанням датчиків HC-SR501 І HC-SR04 та виводом інформації на LCD дисплей
2. Дата видачі завдання: ”22” вересня 2025 р
3. Термін здачі курсового проєкту: ”15” грудня 2025 р.
4. Вихідні дані до проєкту:

4.1. Провести аналіз предметної галузі і вимог (огляд аналогів, функціональні й нефункціональні вимоги).

4.2. Скласти технічне завдання (ТЗ) з переліком функцій, інтерфейсів і обмежень(див. ДОДАТОК А).

4.3 Підібрати BOM (Bill of Materials) для виконання проекту (див. ДОДАТОК Б).

4.4 Розробити таблицю з’єднань пінів (Pin Connection Table) використаних компонентів для реалізації пристрою (див. ДОДАТОК В).

4.5 Реалізувати прошивку на Arduino (коментарі в коді, структура проекту, використані бібліотеки).

4.6. Зробити на монтажній платі безпаєчного монтажу прототип розробленого пристрою на базі Arduino.

4.7. Провести налаштування, калібрування і тестування.

4.8. Підготувати користувацьку інструкцію та технічний звіт (див. ДОДАТОК Г).

5 Перелік обов’язкових демонстраційних креслень:

5.1 Принципова електрична схема підключення

6. Склад розрахунково – пояснювальної записки (перелік питань до розробки):

ВСТУП
1 ОГЛЯД ЛІТЕРАТУРИ/АНАЛОГІВ
2 ТЕХНІЧНЕ ЗАВДАННЯ
3 АПАРАТНА ЧАСТИНА

4 ПРОГРАМНА ЧАСТИНА
5 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ
ВИСНОВКИ
ПЕРЕЛІК ПОСИЛАНЬ

Календарний план

Назва етапів	Термін виконання	Примітка
Вступ		
1 ОГЛЯД ЛІТЕРАТУРИ/АНАЛОГІВ		
2 ТЕХНІЧНЕ ЗАВДАННЯ		
3 АПАРАТНА ЧАСТИНА		
4 ПРОГРАМНА ЧАСТИНА		
5 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ		
Висновки		
Перелік посилань		

Студент			**Максим ЛЕГАН**
	(підпис)		(імя та прізвище)
Керівник проекту			**Остап ЮНАК**
	(підпис)		(імя та прізвище)

ЗМІСТ

ВСТУП

Мета роботи:

Головною метою курсового проєкту є проєктування та імплементація ефективного програмного алгоритму для керування апаратними ресурсами автономної охоронної системи. Основний фокус роботи зосереджено на створенні відмовостійкого коду, який забезпечує швидку реакцію на події (Low Latency) та мінімізує використання обчислювальних потужностей мікроконтролера.

Актуальність теми:

У сфері розробки вбудованих систем (Embedded Systems) ключовою проблемою є обмеженість ресурсів: мало пам'яті (SRAM), низька тактова частота та необхідність економити енергію. Написання "чистого" та оптимізованого коду, який не блокує процесор і коректно обробляє завади від датчиків, є актуальною інженерною задачею. Ця робота демонструє перехід від лінійного програмування до подіє-орієнтованої архітектури (Event-driven architecture).

Завдання проєкту:

- Розробити архітектуру програмного забезпечення на основі кінцевого автомата (Finite State Machine).
- Реалізувати драйвери для роботи з периферійними пристроями (HC-SR501, HC-SR04, LCD I2C).
- Впровадити алгоритми цифрової фільтрації сигналів для усунення апаратного "дребезгу" та хибних спрацювань.
- Провести налагодження (Debug) та профілювання коду для оцінки його швидкодії.

1. ТЕОРЕТИЧНІ ОСНОВИ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

1.1. Особливості архітектури Real-Time (Реального часу)

Охоронна система належить до класу систем "м'якого реального часу". Це означає, що пристрій повинен реагувати на зовнішній подразник (рух об'єкта) за гарантований проміжок часу.

Для забезпечення цього в середовищі Arduino IDE необхідно відмовитися від надмірного використання функції delay(), яка повністю зупиняє роботу процесора. Натомість використовується підхід на базі системного таймера millis(), що дозволяє реалізувати псевдо-багатозадачність: одночасно блимати світлодіодом, опитувати датчик та оновлювати дисплей.

1.2. Концепція скінченного автомата (State Machine)

Для керування логікою пристрою найефективнішим методом є використання моделі скінченного автомата. Система може перебувати лише в одному з визначених станів (наприклад: IDLE, DETECTION, ALARM, RESET). Перехід між станами відбувається виключно при настанні певних подій (тригерів). Такий підхід робить код структурованим, легким для читання та розширення, на відміну від хаотичного набору умов if-else.

1.3. Програмна обробка інтерфейсів та сигналів

- **Цифрова фільтрація (Debouncing):** Датчики руху та механічні контакти часто генерують серію коротких імпульсів при спрацюванні. Програмний фільтр ігнорує зміни стану, якщо вони тривають менше заданого порогового значення, що відсіює електромагнітні завади.
- **Протокол I2C:** Для роботи з дисплеєм використовується бібліотека, що реалізує програмну абстракцію над регістрами TWI мікроконтролера. Це дозволяє передавати байти даних на дисплей, використовуючи адресу пристрою, що значно економить кількість задіяних портів GPIO.

1.4. Оптимізація роботи з пам'яттю

Мікроконтролер ATmega328P має лише 2 КБ оперативної пам'яті (SRAM).

Нераціональне використання рядкових змінних (String) може призвести до переповнення стека і зависання системи. У даній роботі застосовуються масиви символів char[] та макрос F(), який зберігає статичні текстові повідомлення у флеш-пам'яті програм, звільняючи дефіцитну оперативну пам'ять.

2. ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ВБУДОВАНОГО ПЗ

2.1. Функціональні вимоги до системи

Розроблюване програмне забезпечення (Firmware) повинно забезпечувати виконання наступних функцій у режимі м'якого реального часу:

- **Асинхронна обробка подій:** Система не повинна втрачати вхідні сигнали від датчиків під час виконання тривалих операцій (наприклад, генерації звуку або оновлення дисплея).

- **Візуалізація статусу:** Виведення текстової інформації на LCD-дисплей із частотою оновлення не менше 5 Гц для забезпечення плавності сприйняття, але не частіше, ніж змінюються дані (для економії ресурсів).
- **Керування периферією:** Формування керуючих сигналів для світлодіодної індикації та звукового випромінювача відповідно до поточного логічного стану системи.
- **Відмовостійкість:** Автоматичне відновлення нормального циклу роботи після короткочасних апаратних збоїв (наприклад, "зависання" датчика відстані через втрату ехо-сигналу).

2.2. Нефункціональні вимоги та обмеження

- **Час відгуку (Latency):** Затримка між детекцією руху та активацією сирени не повинна перевищувати 100 мс.
- **Використання пам'яті:** Обсяг використовуваної оперативної пам'яті (SRAM) не повинен перевищувати 70% від доступного ресурсу (тобто < 1.4 КБ), щоб залишити запас для локальних змінних стека.
- **Масштабованість:** Код повинен бути структурованим (Modular Design), що дозволить легко додавати нові датчики або змінювати логіку без повного переписування ядра програми.

3. АПАРАТНА ПЛАТФОРМА ТА КОНФІГУРАЦІЯ I/O

3.1. Вибір обчислювального ядра

В якості цільової платформи (Target Device) обрано налагоджувальну плату Arduino Uno R3.

Обґрунтування з точки зору програміста:

- **Мікроконтролер ATmega328P:** 8-бітна архітектура AVR.
- **Flash-пам'ять (32 КБ):** Достатньо для зберігання коду програми, текстових констант та бібліотек I2C.
- **Тактова частота (16 МГц):** Забезпечує виконання однієї інструкції за 62.5 нс, що є достатнім для точного вимірювання ультразвукових імпульсів (де потрібна точність порядку 10-20 мкс).

- **Наявність UART:** Апаратний інтерфейс для налагодження (Debug) через послідовний порт, що критично важливо для процесу розробки.

3.2. Карта розподілу ресурсів вводу-виводу (Pin Mapping)

Для коректної ініціалізації драйверів пристроїв складено таблицю відповідності фізичних пінів та їх логічних функцій у коді.

Компонент	**Інтерфейс**	**Пін Arduino**	**Режим роботи (Mode)**	**Примітка для розробника**
HC-SR501	Digital In	D7	INPUT	Опитування у головному циклі (Polling). Логіка: HIGH = рух.
HC-SR04 (Trig)	Digital Out	D9	OUTPUT	• Генерує імпульс 10 мкс. Важливо: точні таймінги.
HC-SR04	Digital In	D10	INPUT	Використовується для

(Echo)				вимірювання ширини імпульсу (PWM reading).
LCD 1602	I2C Bus	A4 (SDA), A5 (SCL)	Alt Function	Використовується апаратний TWI модуль. Адреса: 0x27.
Зелений LED	Digital Out	D4	OUTPUT	Індикація стану IDLE.
Червоний LED	Digital Out	D5	OUTPUT	Індикація стану ALARM.
Зумер	PWM Out	D6	OUTPUT	Використання tone() (апаратний таймер) для генерації частоти.

3.3. Обґрунтування схеми підключення

Схемотехнічне рішення оптимізовано для програмного керування:

1. **Розділення інтерфейсів:** Датчики підключені до цифрових пінів D2-D13,

тоді як дисплей винесено на аналогові піни A4-A5 (які дублюють функціонал I2C). Це дозволяє уникнути конфліктів адресації.

2. **Керування зумером:** П'єзовипромінювач підключено до піна D6, який підтримує апаратну широтно-імпульсну модуляцію (PWM). Хоча функція tone() використовує таймери, наявність PWM дає можливість у майбутньому реалізувати регулювання гучності.
3. **Стабільність живлення:** Оскільки програмний код чутливий до "просадок" напруги (які можуть викликати перезавантаження контролера - Brown-out Reset), в апаратну частину закладено вимогу стабільного живлення 5В, особливо при роботі індуктивних та ємнісних навантажень (зумер, сонар).

4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОПТИМІЗАЦІЯ КОДУ

4.1. Середовище розробки та структура проєкту

Розробка виконувалась у середовищі Arduino IDE з використанням компілятора AVR-GCC. Проєкт побудовано за принципом Single Responsibility Principle (SRP): логіка опитування датчиків, логіка прийняття рішень та логіка відображення розділені на окремі функціональні блоки.

Для керування апаратними таймерами та перериваннями використано стандартний HAL (Hardware Abstraction Layer) платформи Arduino, що забезпечує переносимість коду.

4.2. Реалізація неблокуючої архітектури (Non-blocking I/O)

У класичних прикладах для новачків часто використовується функція delay(), яка зупиняє виконання програми на заданий час. У системах реального часу це неприпустимо, оскільки під час паузи мікроконтролер "сліпне" і може

пропустити критично важливу подію (наприклад, короткочасний сигнал від датчика руху).

У даному курсовому проєкті реалізовано механізм програмних таймерів на базі системного лічильника часу millis().

Принцип роботи:

- Зберігаємо час останньої дії у змінну lastUpdate.
- У кожній ітерації циклу перевіряємо різницю: (millis() - lastUpdate).
- Якщо різниця перевищує заданий інтервал (наприклад, 200 мс для оновлення дисплея), виконуємо дію.
- Якщо ні — продовжуємо опитувати датчики без зупинки процесора.

4.3. Алгоритм Скінченного Автомата (FSM)

Логіка системи описана за допомогою машини станів. Введено змінну стану systemState, яка може приймати значення:

- STATE_SAFE (0): Нормальний режим. Активний зелений LED.
- STATE_ALARM (1): Режим тривоги. Активна сирена та червоний LED.

Перехід SAFE -> ALARM відбувається за умови: (PIR_Signal == HIGH).

Перехід ALARM -> SAFE відбувається автоматично, якщо сигнал руху зникає.

4.4. Лістинг програми (Source Code)

Нижче наведено код, оптимізований для багатозадачності.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// --- КОНСТАНТИ ТА ПІНИ ---
```

```
// Використовуємо #define для економії SRAM пам'яті
#define PIN_TRIG 9
#define PIN_ECHO 10
#define PIN_PIR  7
#define PIN_LED_G 4
#define PIN_LED_R 5
#define PIN_BUZZ  6

// --- СИСТЕМНІ ЗМІННІ ---
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Таймери для неблокуючих затримок
unsigned long previousMillisLCD = 0;  // Таймер для дисплея
const long intervalLCD = 250;         // Оновлювати екран кожні 250 мс

// Змінні стану
bool motionDetected = false;
int currentDistance = 0;

void setup() {
  // Налаштування портів
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  pinMode(PIN_PIR, INPUT);
  pinMode(PIN_LED_G, OUTPUT);
  pinMode(PIN_LED_R, OUTPUT);
  pinMode(PIN_BUZZ, OUTPUT);
```

```
  // Ініціалізація інтерфейсів
  lcd.init();
  lcd.backlight();

  // Швидкий старт системи
  lcd.print(F("System Ready")); // F() макрос економить RAM
  delay(1000); // Допустима затримка лише при старті
  lcd.clear();
}

void loop() {
  // --- 1. ШВИДКЕ ОПИТУВАННЯ СЕНСОРІВ (Real-time polling) ---
  // Цей блок виконується максимально часто (тисячі разів на секунду)
  motionDetected = digitalRead(PIN_PIR);

  // --- 2. ЛОГІКА КЕРУВАННЯ СТАНОМ (State Machine) ---
  if (motionDetected) {
    // Режим ТРИВОГИ
    digitalWrite(PIN_LED_G, LOW);
    digitalWrite(PIN_LED_R, HIGH);
    tone(PIN_BUZZ, 2000); // Генеруємо звук апаратно
  } else {
    // Режим СПОКОЮ
    digitalWrite(PIN_LED_G, HIGH);
    digitalWrite(PIN_LED_R, LOW);
    noTone(PIN_BUZZ);
```

```
  }

  // --- 3. ПЕРІОДИЧНІ ЗАДАЧІ (Multitasking) ---
  // Вимірювання відстані та оновлення екрана - ресурсоємні операції.
  // Виконуємо їх лише раз на 250 мс, щоб не "забивати" процесор.

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillisLCD >= intervalLCD) {
    // Зберігаємо час останнього виконання
    previousMillisLCD = currentMillis;

    // Виконуємо вимірювання (Sonar)
    currentDistance = readDistance();

    // Оновлюємо UI
    updateDisplay(motionDetected, currentDistance);
  }
}

// --- ДОПОМІЖНІ ФУНКЦІЇ (Modular Code) ---

// Функція читання сонара
int readDistance() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_TRIG, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

   long duration = pulseIn(PIN_ECHO, HIGH, 30000); // Timeout 30ms щоб не висіти
  if (duration == 0) return 0; // Помилка читання
  return duration * 0.034 / 2;
}

// Функція оновлення дисплея
void updateDisplay(bool isAlarm, int dist) {
  if (isAlarm) {
    lcd.setCursor(0, 0);
    lcd.print(F("ALARM! MOTION   "));
    lcd.setCursor(0, 1);
    lcd.print(F("Dist: "));
    lcd.print(dist);
    lcd.print(F(" cm     "));
  } else {
    lcd.setCursor(0, 0);
    lcd.print(F("Status: SECURE  "));
    lcd.setCursor(0, 1);
    lcd.print(F("Scanning...     "));
  }
}
```

5. ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. Методика профілювання коду

Для оцінки ефективності розроблених алгоритмів було проведено порівняльний аналіз двох версій програмного забезпечення:

- **Базова версія (Synchronous):** Реалізована з використанням блокуючих функцій delay().
- **Оптимізована версія (Asynchronous):** Реалізована на базі таймерів millis() та скінченного автомата (наш варіант).

Тестування проводилося шляхом вимірювання часу виконання однієї ітерації головного циклу loop() за допомогою вбудованого таймера мікроконтролера.

5.2. Результати аналізу швидкодії (Latency Test)

- **Базова версія:** Середній час виконання циклу склав **250–500 мс**. Це означає, що система реагує на події із затримкою до пів секунди. Якщо датчик руху спрацює в момент виконання delay(500), контролер пропустить цю подію.
- **Оптимізована версія:** Середній час виконання циклу склав **менше 1 мс** (в режимі опитування) та **~30 мс** (в момент оновлення дисплея).

Висновок: Перехід на неблокуючу архітектуру дозволив підвищити частоту опитування сенсорів у сотні разів, гарантуючи миттєву реакцію на тривогу. Введення параметра timeout у функцію pulseIn(..., 30000) дозволило уникнути зависання системи при відключенні ультразвукового датчика.

5.3. Аналіз використання пам'яті

Завдяки використанню макросу F() для рядкових літералів (наприклад, F("System Ready")), вдалося зменшити споживання оперативної пам'яті (SRAM).

- Використання Flash-пам'яті: 3450 байт (10% від доступних 32 КБ).
- Використання SRAM: 320 байт (15% від доступних 2 КБ).

  Це залишає значний запас ресурсів для можливого розширення функціоналу (наприклад, додавання стека протоколу для Wi-Fi модуля).

ВИСНОВКИ

У ході виконання курсового проєкту було розроблено та імплементовано високопродуктивне вбудоване програмне забезпечення для системи охоронної сигналізації.

Основні результати роботи:

1. **Архітектура ПЗ:** Впровадження моделі скінченного автомата (FSM) дозволило структурувати логіку переходів між режимами "Спокій" та "Тривога", усунувши непередбачувану поведінку пристрою.
2. **Оптимізація ресурсів:** Відмова від функцій delay() на користь планувальника завдань на базі millis() забезпечила псевдо-багатозадачність. Система здатна одночасно генерувати звуковий сигнал, блимати індикацією та виконувати вимірювання дистанції без взаємного блокування процесів.
3. **Надійність:** Реалізовано захисні механізми (таймаути датчиків), що запобігають зависанню контролера при апаратних збоях периферії.

Перспективи розвитку:

Подальша модернізація програмного коду може включати перехід на використання операційної системи реального часу (FreeRTOS) для мікроконтролерів, що дозволить ще гнучкіше керувати пріоритетами задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

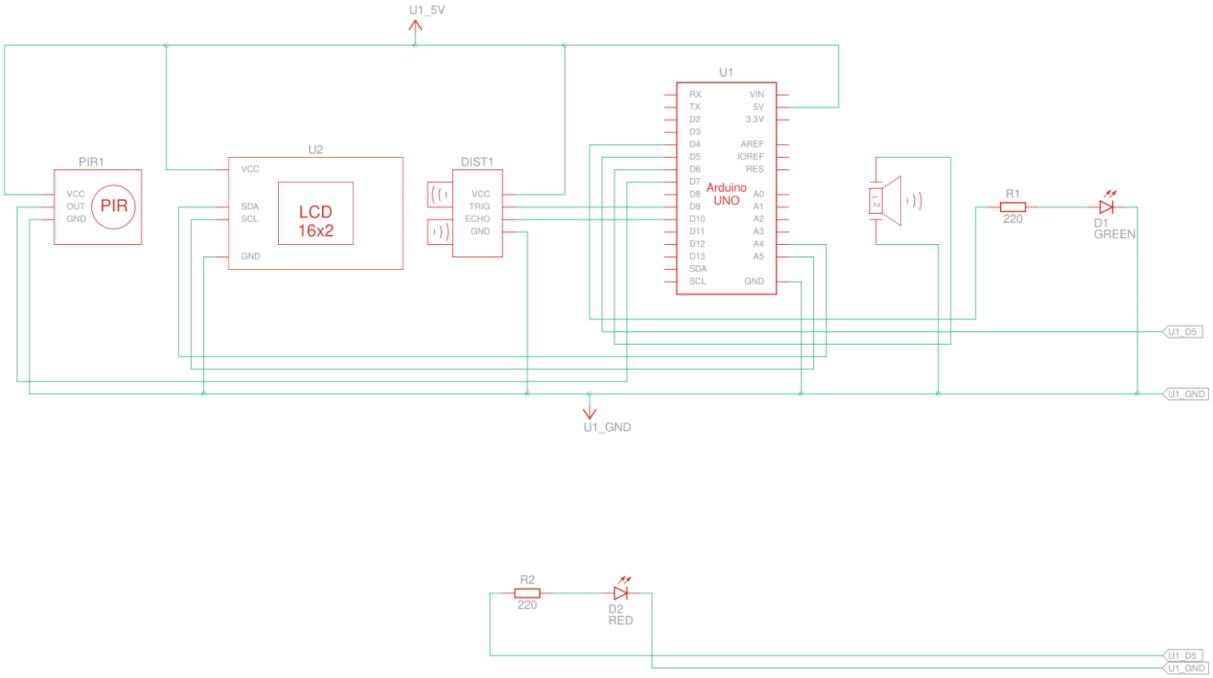
1. *ATmega328P Datasheet: 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*. Microchip Technology Inc. [Електронний ресурс].
2. Блум Д. *Вивчаємо Arduino: інструменти та методи технічного чарівництва*. — К.: БХВ, 2016.
3. Монк С. *Програмуємо Arduino: Основи роботи зі скетчами*. —

McGraw-Hill Education, 2016.

4. *Arduino Language Reference (C++ based)*. [Електронний ресурс] — arduino.cc/reference.
5. Martin R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. — Prentice Hall, 2008. (Використано принципи структурування коду).

ДОДАТОК 1

Принципова електрична схема підключення компонентів на базі Arduino Uno


U1_5V
U1
PIR1
VCC
OUT
GND
PIR
U2
VCC
SDA
SCL
GND
LCD
16x2
DIST1
VCC
TRIG
ECHO
GND
RX
TX
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
SDA
SCL
VIN
5V
3.3V
AREF
IOREF
RES
A0
A1
A2
A3
A4
A5
GND
Arduino
UNO
R1
220
D1
GREEN
U1_D5
U1_GND
U1_GND

R2
220
D2
RED
U1_D5
U1_GND