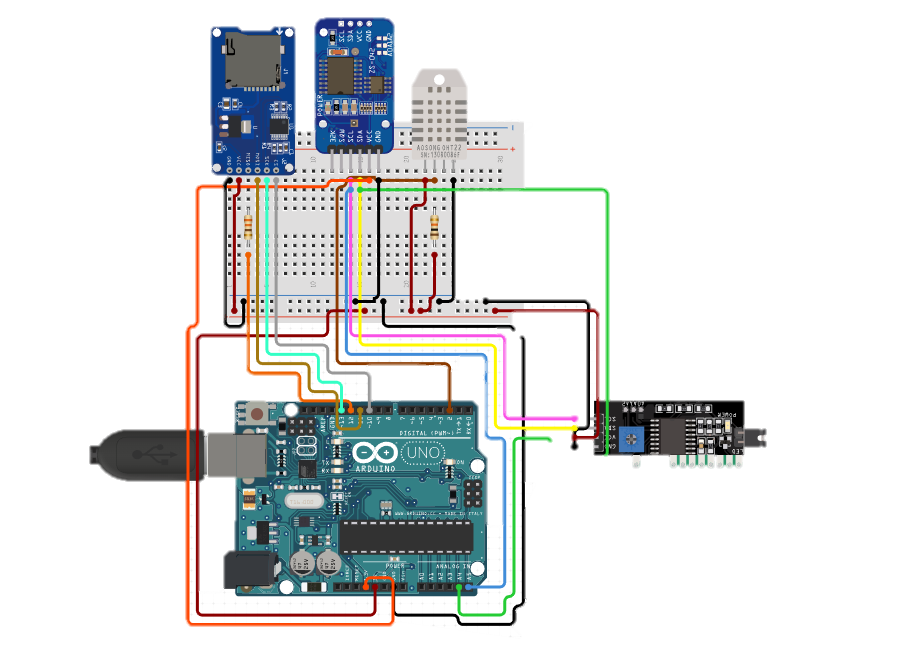
|  |  |
| --- | --- |
| DHT11 온습도 센서 | |
| 회로 구성요소, 전자 공학이(가) 표시된 사진  자동 생성된 설명 |  **전압 요구 사항**: 3.3V ~ 5.5V   **온도 측정 범위**: 0°C ~ 50°C   * **정확도**: ±2°C    **습도 측정 범위**: 20% ~ 90% RH   * **정확도**: ±5% RH    **출력 신호**: 디지털 신호(단일 와이어 통신)   **응답 시간**: 5초   **작동 전류**: 최대 2.5mA   **사이즈**: 15.5mm x 12mm x 5.5mm |

|  |  |
| --- | --- |
| 아두이노 마이크로 SD카드 모듈 DM109 | |
| 전자 부품, 회로 구성요소, 패시브 회로 부품, 전자 공학이(가) 표시된 사진  자동 생성된 설명 |  **전압 요구 사항**: 4.5V ~ 5.5V   **지원 파일 시스템**: FAT16 및 FAT32   **인터페이스**: SPI (Serial Peripheral Interface)   **핀 아웃**:   * CS (Chip Select): 데이터 선택 핀 * MOSI (Master Out Slave In): 데이터 전송 핀 * MISO (Master In Slave Out): 데이터 수신 핀 * SCK (Serial Clock): SPI 클록 핀    **지원 메모리 용량**: 최대 2GB (표준 SD), 최대 32GB (microSD HC)   **사이즈**: 42mm x 24mm |

|  |  |
| --- | --- |
| SEN0219 적외선 CO2 센서 | |
| Infrared CO2 Sensor, 0 to 5000 ppm, Arduino Boards |  **측정 범위**: 400ppm ~ 5000ppm   **전압 요구 사항**: 5V   **출력 방식**:   * 아날로그 출력: 0.4V ~ 2.0V (400ppm ~ 5000ppm) * 디지털 PWM 신호 출력    **정확도**: ±50ppm + 3% (400ppm ~ 5000ppm)   **응답 시간**: ≤30초 (90% 응답 도달)   **작동 온도**: 0°C ~ 50°C   **전력 소비**: <100mW   **사이즈**: 37mm x 17mm x 8mm   **특징**:   * Gravity 인터페이스 지원 * 예열 시간 약 3분 필요 |

|  |  |
| --- | --- |
| RTC(Real Time Clock) 모듈(DS1302) | |
| 전자 공학, 전자제품, 회로 구성요소, 전자 부품이(가) 표시된 사진  자동 생성된 설명 |  **전압 요구 사항**:   * 작동 전압: 2.0V ~ 5.5V * 백업 배터리 전압: 2.0V ~ 3.5V    **시간 정밀도**:   * ±2ppm(25°C에서) → 월별 약 1분 오차    **인터페이스**: 3핀 직렬 통신 (RST, DAT, CLK)   **데이터 보존**: 배터리 백업으로 10년 이상   **특징**:   * 초, 분, 시간, 요일, 날짜, 월, 년도 정보 저장 * Leap Year 자동 보정 * 낮은 전력 소모 *  **사이즈**: 23mm x 14mm x 5mm |

회로도



전자제품, 회로, 전자 공학, 전자 부품이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| 전체 코드 |
| #include "DHT.h"  #include <SPI.h>  #include <SD.h>  #include <Wire.h>  #include <RTClib.h>  #include <avr/wdt.h>  #define DHTPIN 3  #define DHTTYPE DHT11  #define CO2\_THRESHOLD 1000 // CO2 농도 임계값  #define FILENAME\_PREFIX "log\_"  #define JSON\_FORMAT true // JSON 형식 저장 여부  #define BUZZER\_PIN 6  #define LED\_PIN 7  DHT dht(DHTPIN, DHTTYPE);  RTC\_DS3231 rtc;  float lastHumidity = -1;  float lastTemperature = -1;  float lastCO2 = -1;  bool sdInitialized = false;  char currentFileName[20];  String tempDataBuffer = ""; // SD 카드 오류 시 임시 저장 데이터 버퍼  void setup() {  Serial.begin(9600);  dht.begin();  pinMode(BUZZER\_PIN, OUTPUT);  pinMode(LED\_PIN, OUTPUT);  if (!rtc.begin()) {  Serial.println("RTC not found");  while (1);  }  rtc.adjust(DateTime(F(\_\_DATE\_\_), F(\_\_TIME\_\_)));  if (SD.begin(4)) {  sdInitialized = true;  Serial.println("SD initialized");  } else {  sdInitialized = false;  Serial.println("SD failed");  }  updateLogFileName();  }  void loop() {  wdt\_reset(); // 워치독 타이머 리셋  // 온습도 및 CO2 데이터 읽기  float humidity = dht.readHumidity();  float temperature = dht.readTemperature();  if (isnan(humidity) || isnan(temperature)) {  Serial.println("DHT sensor error");  humidity = lastHumidity; // 이전 데이터 사용  temperature = lastTemperature;  } else {  lastHumidity = humidity;  lastTemperature = temperature;  }  float co2 = readCO2Sensor();  if (co2 == -1) {  Serial.println("CO2 sensor error");  co2 = lastCO2; // 이전 데이터 사용  } else {  lastCO2 = co2;  }  // 데이터 검증 및 필터링  if (humidity >= 0 && temperature >= 0 && co2 >= 0) {  String dataString = formatData(humidity, temperature, co2);  saveData(dataString);  checkThreshold(co2);  }  delay(60000); // 1분 간격 측정  }  void saveData(const String& data) {  if (sdInitialized) {  File dataFile = SD.open(currentFileName, FILE\_WRITE);  if (dataFile) {  dataFile.println(data);  dataFile.close();  } else {  Serial.println("SD card error, saving to buffer");  tempDataBuffer += data + "\n";  sdInitialized = false;  }  } else {  tempDataBuffer += data + "\n";  }  }  void updateLogFileName() {  DateTime now = rtc.now();  snprintf(currentFileName, sizeof(currentFileName), "%s%02d%02d%02d.txt",  FILENAME\_PREFIX, now.year() % 100, now.month(), now.day());  }  String formatData(float humidity, float temperature, float co2) {  if (JSON\_FORMAT) {  return String("{\"time\":\"") + rtc.now().timestamp() +  "\",\"humidity\":" + String(humidity, 1) +  ",\"temperature\":" + String(temperature, 1) +  ",\"co2\":" + String(co2, 1) + "}";  } else {  return String(rtc.now().timestamp()) + "," + String(humidity, 1) + "," +  String(temperature, 1) + "," + String(co2, 1);  }  }  void checkThreshold(float co2) {  if (co2 > CO2\_THRESHOLD) {  digitalWrite(LED\_PIN, HIGH);  tone(BUZZER\_PIN, 1000, 500);  Serial.println("CO2 Threshold exceeded!");  } else {  digitalWrite(LED\_PIN, LOW);  noTone(BUZZER\_PIN);  }  }  float readCO2Sensor() {  // CO2 센서 읽기 로직 (예시용)  return random(400, 1200); // 임의 값 반환  } |

|  |
| --- |
| 주요코드 |
| **센서 오류 처리** |
| if (isnan(humidity) || isnan(temperature)) {  humidity = lastHumidity;  temperature = lastTemperature;  }  **DHT 센서 오류 처리**:   * 온도와 습도를 읽지 못하는 경우(NaN 발생), 이전에 읽은 값을 유지하여 시스템의 정상 동작 유지.   CO₂ 데이터를 읽지 못하는 경우(-1 발생), 마지막 유효 데이터를 유지. |
| **2. SD 카드 오류 관리**   * **SD 카드 상태 확인 및 복구**:   + SD 카드 초기화에 실패하면 데이터를 메모리(버퍼)에 임시 저장.   + 카드가 복구되면 버퍼에 저장된 데이터를 기록.   if (!sdInitialized) {  tempDataBuffer += data + "\n";  } else {  File dataFile = SD.open(currentFileName, FILE\_WRITE);  dataFile.println(data);  } |
| **3. 데이터 저장 및 관리**   * **JSON 데이터 포맷 지원**:   + 데이터를 JSON 형식으로 구조화하여 저장.   + 필요 시 CSV 형식으로 전환 가능.   String formatData(float humidity, float temperature, float co2) {  return String("{\"time\":\"") + rtc.now().timestamp() +  "\",\"humidity\":" + String(humidity, 1) +  ",\"temperature\":" + String(temperature, 1) +  ",\"co2\":" + String(co2, 1) + "}";  }  **날짜 기반 파일 관리**:   * 매일의 데이터를 log\_YYYYMMDD.txt 형식으로 저장하여 관리 용이.   snprintf(currentFileName, sizeof(currentFileName), "%s%02d%02d%02d.txt",  FILENAME\_PREFIX, now.year() % 100, now.month(), now.day());  4. **전력 효율 및 워치독 타이머**   **전력 효율**:   * 저전력 모드를 구현할 수 있는 기반 제공(센서를 필요한 경우에만 활성화).    **워치독 타이머**:   * 8초 간격으로 시스템 상태를 리셋하여 무한 루프나 오류로 인한 시스템 정지 방지.   #include <avr/wdt.h>    wdt\_enable(WDTO\_8S);  wdt\_reset() |