BAB IV
PEMBUATAN ALAT

4.1 Pembuatan Prototipe

Dalam proses perancangan dan implementasi proyek akhir ini, diperlukan suatu metode penelitian yang sistematis guna memastikan konsep yang matang serta struktur yang jelas dalam pengembangan sistem. Bab 4 menguraikan tahapan perancangan dan pembuatan perangkat, mencakup seluruh proses mulai dari perancangan awal, pemilihan komponen, hingga implementasi dan pengujian agar perangkat dapat beroperasi sesuai dengan spesifikasi yang telah ditentukan. Setiap tahapan dalam pengembangan sistem memiliki keterkaitan erat dalam membentuk solusi yang terintegrasi. Secara umum, prosedur pembuatan perangkat dalam tugas akhir ini terbagi menjadi dua tahapan utama, yaitu:

a. Pembuatan Perangkat Keras

Perangkat keras merupakan komponen fisik yang digunakna untuk mengumpulkan data dan mengirimkannya ke cloud melalui koneksi internet. Pembuatan perangkat keras melalui proses identifikasi dan penyediaan kebutuhan perangkat keras menjadi proses awal untuk memastikan bahwa desain alat dapat diimplementasikan secara optimal. Selanjutnya pembuatan perangkat keras, yang bertujuan untuk merealisasikan rancangan menjadi suatu sistem yang siap beroperasi.

b. Pembuatan Perangkat Lunak

Tahap selanjutnya adalah pembuatan perangkat lunak yang berfungsi untuk mengatur alur kerja dari perangkat keras, mengolah data sensor, serta menghubungkan perangkat dengan Arduino IOT Cloud untuk pemanfaatan data logger. Langkah awal dalam menyelesaikan tugas akhir adalah dengan menyusun perencanaan yang matang serta merumuskan konsep yang jelas terkait perangkat atau produk yang akan dikembangkan. Tahapan ini sangat penting karena akan menjadi dasar bagi keseluruhan proses, mulai dari analisis kebutuhan hingga implementasi. Pengembangan perangkat lunak dalam sistem ini melibatkan pemrograman menggunakan Arduino IDE.

4.2 Alat dan Bahan

Dalam menjalankan proyek tugas akhir, ada berbagai kebutuhan utama dan pendukung yang harus dipenuhi agar pengembangan berjalan optimal. Identifikasi kebutuhan ini menjadi langkah krusial karena akan menghasilkan daftar komprehensif mengenai komponen dan sumber daya yang diperlukan.

Tabel 4.1 memperlihatkan daftar kebutuhan tersebut, berperan sebagai landasan dalam perancangan dan pembuatan perangkat, memastikan sistem yang dikembangkan sesuai dengan spesifikasi teknis serta tujuan penelitian. Dengan identifikasi yang jelas, risiko kesalahan dalam pemilihan komponen atau pengalokasian sumber daya dapat diminimalkan, sehingga proyek dapat berjalan lebih efisien dan sesuai dengan rencana.

Tabel 4. 1 Bahan Pembuatan Tugas Akhir

NO	KOMPONEN	SPESIFIKASI	JUMLAH
1	Mikrokontroler	ESP32 DEVKIT V1	1
2	Sensor Gas NH3 dan CO2	MQ 135	1
3	Sensor PM 2.5	DSM 501A	1
4	Sensor Suhu & Kelembapan	DHT22	1
5	Modul RTC	RTC DS3231	1
6	Display	DMD P10	4
7	Power Supply	PSU 5V	1
8	Switch	Switch ON/OFF	1
9	Kabel	JST XH 2,54 mm	4
10	Kabel	IDC P10 8 x 2	4
11	Kabel	Kabel Steker	1
12	Terminal Block	Pitch 5.08 mm	2
13	PCB	Fiber	1
14	Frame	Frame 64 x 32	1

Tabel 4. 2 Alat yang digunakan

NO	Nama Alat	Jumlah
1	Multimeter	1
2	Solder	1
3	Penyedot Timah	1
4	Tang Potong	1
5	Bor tangan	1
6	Obeng (-)	1

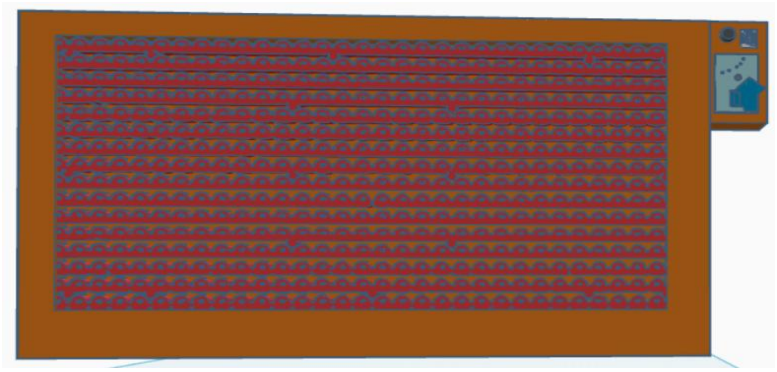
4.3 Pembuatan Perangkat Keras

Proses pembuatan perangkat keras diawali dengan pengumpulan literatur dan referensi dari sumber-sumber terpercaya, seperti buku, jurnal ilmiah, serta studi terkait produk serupa yang telah dikembangkan sebelumnya. Tahap ini dilanjutkan dengan analisis mendalam untuk memahami prinsip kerja sistem, mengidentifikasi kebutuhan komponen seperti jenis sensor, mikrokontroler, modul komunikasi, dan sumber daya listrik yang diperlukan. Selanjutnya, dilakukan perancangan rangkaian elektronik dengan mempertimbangkan koneksi antar komponen secara optimal. Proses ini juga melibatkan pemilihan board mikrokontroler yang sesuai dengan kebutuhan aplikasi, serta modul pendukung seperti sensor dan aktuator yang akan digunakan. Setelah perancangan selesai, tahap berikutnya adalah perakitan perangkat keras dengan menggunakan breadboard atau PCB, pengujian koneksi antar komponen, dan verifikasi fungsi dasar perangkat agar dapat berjalan sesuai dengan spesifikasi yang diinginkan.

4.3.1 Perancangan Mekanik

Setelah tahap pengmbangan perangkat keras selesai dan komponen elektronik terintegrasi dengan baik, langkah selanjutnya adalah melakukan perancangan mekanik. Perancangan mekanik bertujuan untuk merancang struktur fisik atau casing yang dapat melindungi perangkat keras dari gangguan lingkungan, memudahkan pemasangan sensor, serta memastikan stabilitas dan keandalan sistem secara keseluruhan. Dengan demikian, integrasi antara perangkat keras elektronik

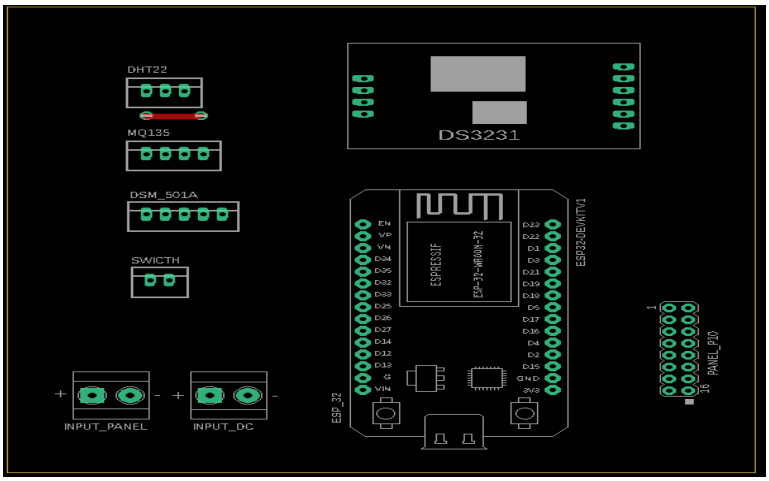
dan rancangan mekanik yang tepat akan menghasilkan produk yang tidak hanya fungsional tetapi juga tahan lama dan mudah digunakan dalam aplikasi nyata.



Gambar 4. 1 Perancangan Frame Alat

4.3.2 Perancangan Alat

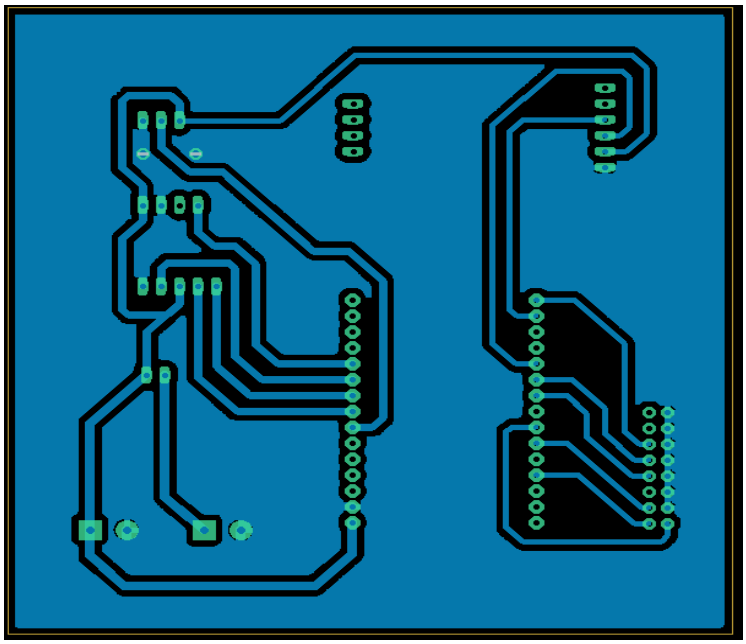
Setelah tahap pembuatan perangkat keras selesai, langkah selanjutnya adalah melakukan perancangan alat secara menyeluruh yang mencakup integrasi komponen elektronik dengan aspek fungsional dan struktural alat tersebut. Perancangan alat bertujuan untuk memastikan bahwa perangkat keras yang telah dikembangkan dapat bekerja secara optimal dalam lingkungan operasionalnya, serta memenuhi kebutuhan pengguna dari segi performa, kemudahan penggunaan, dan keandalan sistem secara keseluruhan. Gambar 4.2 memperlihatkan posisi dari komponen yang akan menjadi acuan dalam pencetakan PCB.



Gambar 4. 2 Posisi Penempatan Komponen

4.3.3 Perancangan PCB

Setelah proses pembuatan perangkat keras dan perancangan alat selesai, langkah berikutnya adalah melakukan perancangan PCB (Printed Circuit Board) sebagai media untuk mengintegrasikan semua komponen elektronik secara rapi dan efisien. Perancangan PCB bertujuan untuk menggantikan rangkaian prototipe sementara seperti breadboard dengan solusi yang lebih stabil, kompak, dan mudah diproduksi. Dengan perancangan PCB yang tepat, perangkat keras dan alat yang telah dirancang dapat diwujudkan dalam bentuk produk akhir yang memiliki kualitas koneksi listrik optimal serta kemudahan dalam perakitan dan pemeliharaan. Gambar 4.3 merupakan rancangan PCB yang akan di cetak.



Gambar 4. 3 Rancangan PCB

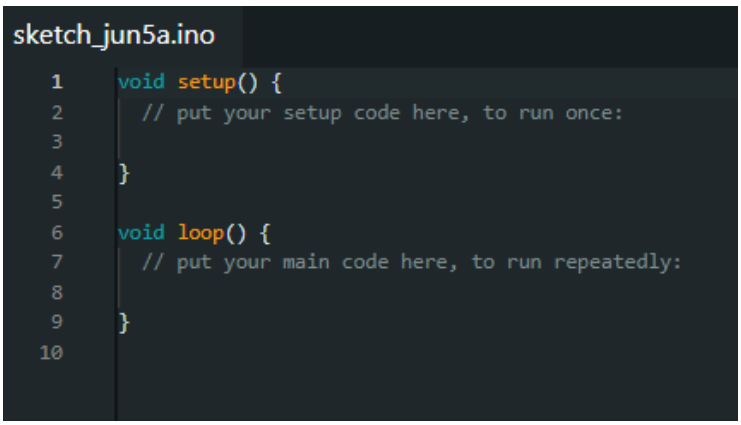
4.4 Pembuatan Perangkat Lunak

Pada tahap ini dilakukan pengembangan perangkat lunak yang melibatkan perancangan dan penulisan kode program menggunakan Arduino IDE. Pengembangan ini merupakan implementasi algoritma sistem pada ESP32 DevKit V1. Pengembangan ini dilakukan agar sistem dapat berjalan sesuai dengan perencanaan. Dengan demikian, perangkat lunak yang dibuat mampu mengendalikan perangkat keras secara optimal serta menjalankan fungsi

pengukuran, pengolahan data, dan komunikasi sesuai kebutuhan aplikasi.

c. Persiapan Program

Langkah ini merupakan proses awal pembuatan sistem pemantauan kualitas udara khususnya CO2, NH3, PM 2.5, suhu dan kelembapan dengan tampilan DMD P10. Sebelum memulai, persiapkan perangkat seperti PC atau Laptop, Kabel USB untuk menghubungkan mikrokontroler ESP32 DevKit V1 dengan computer. Pastikan sudah melakukan isntalasi board dan library yang dibutuhkan.



Gambar 4. 4 Tampilan Awal Arduino IDE

Gambar 4.4 merupakan tampilan awal dari aduino IDE. Sebelum penulisan kode program sistem pemantauan kualitas udara, library yang dibutuhkan untuk di install berupa *PxMatrix* merupakan library dari DMD P10, *RTClib* merupakan library untuk membaca waktu dari modul RTC DS3231, *Arduino IOT Cloud* merupakan library untuk penggunaan platform Arduino IOT Cloud, *DHT* merupakan library yang membaca suhu dan kelembapan dari sensor DHT22, *Arduino Connection Handler, Avarage Value*. Setelah semua library dan board berhasil terinstal, tahap selanjutnya adalah pengujian dasar terhadap masing – masing komponen secara terpisah, dan proses juga proses kalibrasi sensor ataupun komponen yang membutuhkan kalibrasi.

d. Kalibrasi MQ 135 untuk Gas CO2

```ino
#include <AverageValue.h>

// ====== Konfigurasi Pin & Sensor ======
#define PIN_MQ_A0 35  // Pin analog sensor MQ-135 terhubung ke GPIO 35

// ====== Nilai Kalibrasi & Parameter Sensor ======
const int RL = 15000;          // Nilai resistor pull-down (ohm)
const int KALIBRASI_PPM = 400;  // Nilai ppm saat kalibrasi (misal udara bersih)

// ====== Koefisien Persamaan Korelasi ======
// const float a = 110.7432567;      // Koefisien a
// const float b = -2.856935538;     // Koefisien b

const float a = 116.6020682;
const float b = -2.769034857;

const long dataAverage = 10;
AverageValue<long> averagePPM(dataAverage);

bool sudahKalibrasi = false;
float nilaiRo = 0;

void setup() {
  Serial.begin(115200);
  pinMode(PIN_MQ_A0, INPUT);
  analogReadResolution(12);  // Resolusi pembacaan ADC 12-bit (0-4095)
}
```

```ino
void loop() {
  float adc = analogRead(PIN_MQ_A0);

  if (adc == 0) {
    Serial.println("ADC error: nilai 0 tidak valid.");
    delay(1000);
    return;
  }

  // Hitung Rs dari nilai ADC
  float rS = ((4096.0 * RL) / adc) - RL;

  // Proses kalibrasi: hitung Ro sekali
  if (!sudahKalibrasi) {
    nilaiRo = rS * exp(log(a / KALIBRASI_PPM) / b);
    sudahKalibrasi = true;

    // Serial.print("Kalibrasi selesai. Ro = ");
    // Serial.print(nilaiRo);
    // Serial.println(" ohm");
  } else {
    // Hitung PPM berdasarkan Ro
    float ppm = a * pow((float)rS / nilaiRo, b);
    averagePPM.push(ppm);

    float ppmCO2 = averagePPM.average();
    // if (ppmCO2 < 400) { ppmCO2 = 400; }
    if (ppmCO2 > 1000) { ppmCO2 = 1000; }

    Serial.print("Ro = ");
    Serial.print(nilaiRo);
    Serial.print(" ohm, PPM = ");
    Serial.println(ppmCO2);
  }
  delay(1000);
}
```

Gambar 4. 5 Kalibrasi MQ 135 untuk Gas CO2

Gambar 4.5 merupakan program kalibrasi MQ135 untuk Gas CO2, kode program tersebut merupakan implementasi kalibrasi dan pembacaan sensor gas MQ-135 untuk mengukur konsentrasi CO_2 dalam satuan ppm (parts per million). Pada awal program, sensor dihubungkan ke pin analog GPIO 35 pada mikrokontroler ESP32 dengan resolusi ADC 12-bit untuk pembacaan nilai analog dari sensor. Proses kalibrasi dilakukan sekali dengan menghitung nilai resistansi dasar (Ro) sensor pada kondisi udara bersih yang didapat oleh pengukuran CO2 oleh alat industri, konsentrasi CO_2 yang terdeteksi sebesar 400 ppm. Nilai resistansi sensor (Rs) dihitung dari pembacaan ADC dan resistor pull-down yang digunakan dalam rangkaian. Selanjutnya, nilai Ro digunakan sebagai referensi untuk menghitung konsentrasi CO_2 aktual berdasarkan persamaan korelasi dengan koefisien a dan b yang telah ditentukan. Nilai a dan b didapat dari grafik pada datasheet MQ135, grafik tersebut adalah acuan untuk mengkalibrasi sensor agar bisa menemukan nilai ppm. Untuk mencari nilai Rs/ Ro perlu mencari nilai Rs dan nilai Ro. Dimana Rs adalah nilai resistansi Sensor pada konsentrasi gas dan Ro

adalah tahanan sensor pada udara yang bersih. Rs/ Ro juga bisa disebut sebagai rasio. Untuk meningkatkan keakuratan pembacaan, nilai ppm yang dihasilkan dari beberapa pengukuran dirata-ratakan menggunakan AverageValue. Hasil akhir berupa nilai Ro dan konsentrasi CO_2 rata-rata ditampilkan secara berkala melalui serial monitor. Program ini memastikan pembacaan yang stabil dan valid dengan melakukan pengecekan error pada nilai ADC dan membatasi nilai ppm maksimal agar tetap dalam rentang yang realistis, hal ini dikarenakan rentang baca MQ135 terhadap gas CO2 pada datasheet hanya sampai 1000 PPM.

e. Kalibrasi MQ135 Untuk gas NH3

```ino
const int sensorPin = 35;  // Pin ADC untuk sensor MQ
const float RL = 15.0;     // RL (kΩ), resistor beban
const float Vcc = 3.3;     // Tegangan catu daya sensor
const float Ro = 57.0;     // Nilai Ro yang sudah dikalibrasi (kΩ)

// Nilai m dan b diambil dari datasheet MQ-135 (perkiraan)
const float m = -0.417;
const float b = 0.858;

float VRL;
float RS;
float RO;
float MAX;
float ppmNH3;

void setup() {
  Serial.begin(115200);
}

void loop() {
  int adcValue = analogRead(sensorPin);
  float Vrl = adcValue * (Vcc / 4096.0);  // Tegangan di RL
  float Rs = (Vcc - Vrl) * RL / Vrl;      // Hitung Rs
  float ratio = Rs / Ro;                  // Rasio Rs/Ro
```

```ino
  // //Rumus mencari RO
  // VRL = adc * (3.3 / 4096.0);
  // RS = ((3.3 / VRL) - 1) * 10;
  // RO = RS / 3.6;
  // if (RO > MAX) MAX = RO;

  // Serial.print("RO:");
  // Serial.println(MAX);
  // delay(1000);

  // Hitung ppm NH3 menggunakan persamaan logaritmik
  float ppmNH3 = pow(10, (log10(ratio) - b) / m);

  // Tampilkan hasil
  Serial.print("ADC: ");
  Serial.print(adcValue);
  Serial.print(" | Rs: ");
  Serial.print(Rs);
  Serial.print(" kOhm | Ratio: ");
  Serial.print(ratio);
  Serial.print(" | NH3 ppm: ");
  Serial.println(ppmNH3);

  delay(1000);  // Delay 1 detik
}
```

Gambar 4. 6 Kalibrasi MQ135 untuk gas NH3

Gambar 4.6 merupakan program kalibrasi MQ135 untuk Gas NH3. Kode program di atas digunakan untuk mengukur konsentrasi gas amonia (NH_3) menggunakan sensor gas MQ-135 yang terhubung ke pin ADC 35 pada mikrokontroler ESP32. Program ini memulai dengan membaca nilai analog dari sensor, kemudian mengonversi nilai ADC menjadi tegangan pada resistor beban (RL). Selanjutnya, nilai resistansi sensor (Rs) dihitung berdasarkan tegangan tersebut dan nilai resistor beban yang diketahui. Rasio antara Rs dan nilai resistansi kalibrasi (Ro) digunakan sebagai parameter utama dalam perhitungan konsentrasi NH_3. Nilai ppm NH_3 dihitung dengan menggunakan persamaan logaritmik yang

melibatkan koefisien m dan b, yang diperoleh dari datasheet sensor MQ-135 sebagai pendekatan karakteristik sensor terhadap gas amonia. Hasil pembacaan berupa nilai ADC, resistansi sensor, rasio Rs/Ro, dan konsentrasi NH_3 dalam ppm ditampilkan secara berkala melalui serial monitor. Program ini bertujuan memberikan estimasi kadar NH_3 secara real-time dengan memperhatikan kalibrasi awal dan karakteristik sensor, sehingga dapat digunakan untuk monitoring kualitas udara terkait gas amonia.

f. Konfigurasi Sensor MQ 135

```
void MQ135_begin() {
  Serial.println("Initializing MQ-135 Sensor...");
  pinMode(PIN_MQ_A0, INPUT);
  analogReadResolution(12);  // ADC 12-bit (0–4095)
}

float get_nh3_mq135() {
  static AverageValue<float> averageNH3(dataAverageCount);  // Dijadikan static lokal, nama tetap
  int adcValue = analogRead(PIN_MQ_A0);
  if (adcValue == 0) return nh3;  // Kembalikan nilai lama jika error

  float Vrl = adcValue * (Vcc_NH3 / 4095.0f);  // Gunakan 4095.0f untuk float division
  if (Vrl == 0) return nh3;                    // Hindari pembagian dengan nol

  float Rs = (Vcc_NH3 - Vrl) * RL_NH3 / Vrl;
  float ratio_mq = Rs / Ro_NH3;  // Ganti nama lokal 'ratio' agar tidak konflik dengan global (yang sudah dihapus)

  if (ratio_mq <= 0) return nh3;  // log10 dari non-positif tidak valid

  float ppmNH3 = pow(10, (log10(ratio_mq) - b_NH3_const) / m_NH3);
  averageNH3.push(ppmNH3);  // Menggunakan objek static lokal averageNH3
  float avg_nh3 = averageNH3.average();
  return avg_nh3 > 0 ? avg_nh3 : 0;
}
```

```
float get_co2_mq135() {
  static AverageValue<float> averageCO2(dataAverageCount);  // Dijadikan static lokal, nama tet
  float adc = analogRead(PIN_MQ_A0);
  if (adc == 0) return co2;  // Kembalikan nilai lama jika error

  float rS = ((4095.0f * mqR_CO2) / adc) - mqR_CO2;  // Gunakan 4095.0f
  if (nilaiRo_CO2 == 0) return co2;                  // Hindari pembagian dengan nol jika Ro ti

  float ppm = a_CO2 * pow((float)rS / nilaiRo_CO2, b_CO2);
  averageCO2.push(ppm);  // Menggunakan objek static lokal averageCO2

  float ppmCO2 = averageCO2.average();
  if (ppmCO2 < 400) { ppmCO2 = 400; }  // Batas bawah PPM
  if (ppmCO2 > 999) { ppmCO2 = 999; }  // Batas atas PPM (sesuaikan)
  return ppmCO2;
}
```

Gambar 4. 7 Konfigurasi Sensor MQ135

Gambar 4.7 merupakan kode program yang berisi dua fungsi utama untuk membaca dan memproses data gas NH_3 dan CO_2 menggunakan sensor MQ-135 yang terhubung ke pin analog pada mikrokontroler ESP32. Fungsi *MQ135_begin()* bertugas menginisialisasi sensor dengan mengatur pin input dan resolusi ADC menjadi 12-bit untuk pembacaan yang lebih presisi. Fungsi *get_nh3_mq135()* membaca nilai analog dari sensor, menghitung resistansi sensor (Rs) berdasarkan tegangan pada resistor beban (RL), kemudian menghitung rasio Rs terhadap nilai resistansi kalibrasi (Ro). Dengan menggunakan persamaan logaritmik yang melibatkan koefisien *m_NH3 dan b_NH3_const*, fungsi ini menghitung konsentrasi NH_3 dalam ppm dan menerapkan rata-rata bergerak (moving average) untuk menghasilkan nilai yang lebih stabil dan akurat. Fungsi *get_co2_mq135()* bekerja

serupa, namun menggunakan parameter dan konstanta yang berbeda (a_CO2, b_CO2, nilaiRo_CO2) untuk menghitung konsentrasi CO_2 berdasarkan rasio resistansi sensor terhadap nilai Ro yang sudah dikalibrasi. Nilai ppm CO_2 yang diperoleh juga dirata-ratakan dan dibatasi pada rentang minimal 400 ppm dan maksimal 999 ppm, hal ini dikarenakan penyesuaian batas bacaan dimulai 400 PPM oleh pembacaan alat industry, mengapa dibatas pada 999 PPM hal ini dikarenakan MQ135 hanya dapaat mendeteksi CO2 hingga 1000 PPM saja, dan agar hasil pembacaan tetap realistis. Kedua fungsi ini mengembalikan nilai gas yang terukur secara real-time dan telah difilter untuk mengurangi fluktuasi pembacaan, sehingga cocok digunakan dalam sistem monitoring kualitas udara berbasis ESP32.

g. Konfigurasi Sensor DSM501A

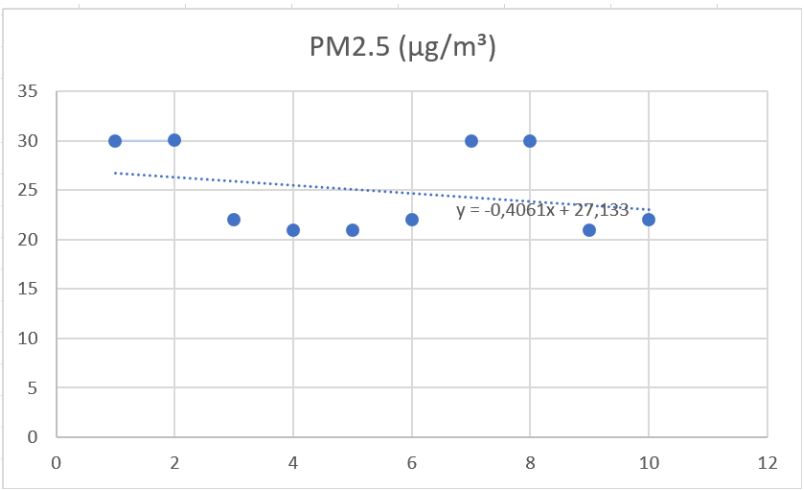
Untuk mendukung pemahaman terhadap kinerja sensor dalam mendeteksi kualitas udara khususnya PM2.5 dibutuhkan nilai dari hasil kalibasi, data yang diperoleh telah diolah dan divisualisasikan dalam bentuk tabel dan grafik berikut ini.

Tabel 4. 3 Tabel Kalibrasi Sensor DSM501A terhadap Nilai PM2.5

Ratio local (%)	**PM2.5 (µg/m³)**
3.48	30
1.39	30
2.14	22
2.09	21
2.40	21
1.08	22
0.18	30
0.86	30
1.02	21
1.78	22

Tabel dan grafik berikut menyajikan hasil pemrosesan data dari sensor DSM501A yang digunakan dalam sistem pemantauan kualitas udara berbasis mikrokontroler. Sensor ini mendeteksi partikel halus (aerosol) di udara dan menghasilkan sinyal digital yang dapat diinterpretasikan sebagai indikator tingkat konsentrasi PM2.5.

Kolom pertama pada tabel menunjukkan nilai rasio_local (%), yaitu persentase waktu sinyal dari sensor berada dalam kondisi LOW terhadap total waktu pengambilan sampel. Rasio ini berkorelasi langsung dengan jumlah partikel yang terdeteksi oleh sensor selama periode pengukuran. Kolom kedua menyajikan nilai konsentrasi PM2.5 (dalam satuan µg/m³) yang diperoleh dari sistem berdasarkan data sinyal tersebut.



Gambar 4. 8 Kurva Linear Kalibrasi PM2.5

Grafik 4.8 yang menyertai tabel 4.3 memvisualisasikan hubungan antara rasio local dan nilai PM2.5. Titik-titik data diplot dalam diagram kartesius, dan garis regresi linear ditambahkan untuk mengidentifikasi pola hubungan antara keduanya. Hasil regresi menghasilkan persamaan: PM2.5 = -0,4061 × ratio_local + 27,133. Persamaan ini digunakan sebagai pendekatan matematis untuk mengestimasi konsentrasi PM2.5 dari keluaran sinyal sensor. Pendekatan regresi tetap relevan untuk memberikan estimasi awal yang cukup representatif. Model ini menunjukkan bahwa sistem mampu menangkap variasi kualitas udara secara real-time khususnya PM2.5.

```
void DSM501A_begin() {
  Serial.println("Initializing DSM501A (PM2.5 Sensor)...");  // Placeholder dihilangkan
  pinMode(PIN_DSM501, INPUT);
}

// Fungsi DSM501A_read diubah untuk me-return float, global lastPM25 dihilangkan
float DSM501A_read() {
  unsigned long duration_local;              // Variabel lokal, nama tidak konflik
  unsigned long lowPulseOccupancy_local = 0;  // Variabel lokal
  unsigned long startTime = millis();

  while (millis() - startTime < samplingTime) {
    duration_local = pulseInLong(PIN_DSM501, LOW, 100000);  // Timeout 100ms
    lowPulseOccupancy_local += duration_local;
    vTaskDelay(20 / portTICK_PERIOD_MS);  // jangan terlalu intens, beri kesempatan task lain
  }

  float ratio_local = (lowPulseOccupancy_local / (samplingTime * 10.0f));  // Gunakan .0f untuk float context
  // Batasi nilai LPO percentage antara 0% dan 100% (jika 'ratio' ini memang persentase)
  // Perlu validasi apakah perhitungan 'ratio' ini sudah benar menghasilkan persentase untuk formula di bawah
  if (ratio_local > 100.0f) {
    ratio_local = 100.0f;
  }
  if (ratio_local < 0.0f) {
    ratio_local = 0.0f;
  }
  // float concentration = 1.1 * pow(ratio_local, 3) - 3.8 * pow(ratio_local, 2) + 520 * ratio_local + 0.62; // Formula alternatif
  //return (20.887f * ratio_local) + 20.667f;  // Langsung return hasil PM2.5
  return (-0.4061f * ratio_local) + 27.133f;  // Langsung return hasil PM2.5
 // return ratio_local;
}
```

Gambar 4. 9 Konfigurasi Sensor DSM501A

Gambar 4.8 merupakan kode program implementasi fungsi untuk membaca konsentrasi partikel debu PM2.5 menggunakan sensor DSM501A yang terhubung ke mikrokontroler ESP32. Fungsi *DSM501A_begin()* bertugas menginisialisasi pin input sensor, sedangkan fungsi *DSM501A_read()* melakukan pengukuran durasi pulsa rendah (low pulse occupancy) dari sensor selama periode sampling tertentu. Durasi pulsa rendah ini diakumulasi dan digunakan untuk menghitung rasio waktu pulsa rendah terhadap total waktu sampling, yang kemudian dikonversi menjadi estimasi konsentrasi PM2.5 menggunakan persamaan linear. Fungsi ini juga membatasi nilai rasio agar tetap berada dalam rentang 0% hingga 100% untuk menjaga validitas perhitungan. Selain itu, penggunaan vTaskDelay memberikan jeda agar tugas lain dalam sistem FreeRTOS dapat berjalan, sehingga pembacaan sensor tidak mengganggu performa keseluruhan mikrokontroler. Hasil pengukuran PM2.5 dikembalikan sebagai nilai float yang merepresentasikan konsentrasi partikel debu dalam satuan mikrogram per meter kubik (µg/m³), yang dapat digunakan untuk pemantauan kualitas udara secara real-time.

h. Konfigurasi Sensor DHT22

```
void DHT22_begin() {
  Serial.println("Initializing DHT22 Sensor...");
  dhtSensor.setup(PIN_DHT22, DHTesp::DHT22);
}

float get_humidity_dht22() {
  TempAndHumidity data = dhtSensor.getTempAndHumidity();
  float humi_val = data.humidity;

  if (isnan(humi_val)) {
    Serial.println("Failed to read humidity from DHT sensor!");
    return -1;  // Indikasi error
  }
  if (humi_val >= 99.0) {
    humi_val = 99.0;
  }
  return humi_val;
}

float get_temperature_dht22() {
  TempAndHumidity data = dhtSensor.getTempAndHumidity();
  float temp_val = data.temperature;

  if (isnan(temp_val)) {
    Serial.println("Failed to read temperature from DHT sensor!");
    return -1;  // Indikasi error
  }
  if (temp_val >= 99.0) {
    temp_val = 99.0;
  }
  return temp_val;
}
```

Gambar 4. 10 Konfigurasi Sensor DHT22

Gambar 4.9 merupakan kode program yang mengimplementasikan fungsi untuk inisialisasi dan pembacaan sensor DHT22 yang digunakan untuk mengukur suhu dan kelembapan udara. Fungsi *DHT22_begin()* bertugas menginisialisasi sensor dengan mengatur pin yang terhubung dan tipe sensor menggunakan library DHTesp. Fungsi *get_humidity_dht22()* membaca nilai kelembapan dari sensor dan melakukan pengecekan validitas data; jika pembacaan gagal (nilai NaN), fungsi akan mengembalikan nilai -1 sebagai indikator error dan menampilkan pesan kesalahan di serial monitor.

Selain itu, nilai kelembapan dibatasi maksimum hingga 99% untuk menghindari pembacaan yang tidak realistis. Fungsi *get_temperature_dht22()* bekerja serupa, begitu pula untuk pembacaan suhu udara dengan batas maksimal 99°C, dan mekanisme penanganan error yang sama. Dengan demikian, program ini memastikan pembacaan data suhu dan kelembapan yang akurat dan stabil dari sensor DHT22, yang sangat penting dalam aplikasi monitoring kualitas udara berbasis mikrokontroler seperti ESP32.

i. Konfigurasi RTC DS3231

```
void display_jam() {
  int jam_disp, menit_disp, detik_disp;

  jam_disp = currentTime.jam;
  menit_disp = currentTime.menit;
  detik_disp = currentTime.detik;

  if (jam_disp < 10) display.print("0");
  display.print(jam_disp);
  display.print(":");
  if (menit_disp < 10) display.print("0");
  display.print(menit_disp);
  display.print(":");
  if (detik_disp < 10) display.print("0");
  display.print(detik_disp);
}

void RTC_begin() {
  Serial.println("Initializing RTC...");
  if (!rtc.begin()) {
    Serial.println("RTC tidak terdeteksi!");
    // while (1) delay(1); // Hentikan jika RTC kritis
  }

  if (rtc.lostPower()) {
    Serial.println("RTC kehilangan daya, mengatur ulang waktu...");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}

float RTC_update_time_and_get_temp() {
  DateTime now = rtc.now();
  currentTime.tahun = now.year();
  currentTime.bulan = now.month();
  currentTime.hari = now.day();
  currentTime.jam = now.hour();
  currentTime.menit = now.minute();
  currentTime.detik = now.second();
  return rtc.getTemperature();
}
```

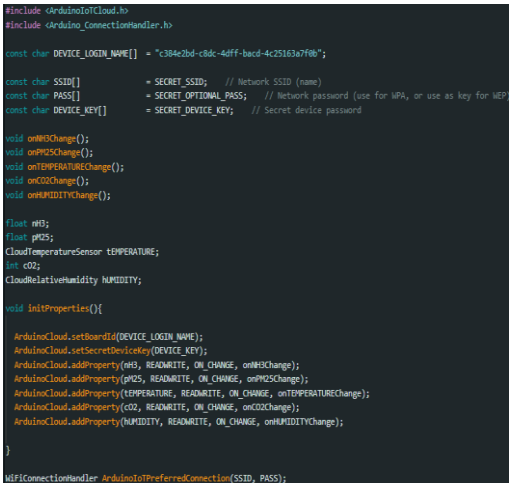
Gambar 4. 11 Konfigurasi RTC DS3231

Gambar 4.10 merupakan kode program yang terdiri dari tiga fungsi utama yang berhubungan dengan pengelolaan waktu dan tampilan waktu menggunakan modul RTC (Real Time Clock) dan display. Fungsi *display_jam()* bertugas menampilkan waktu dalam format jam:menit:detik pada perangkat display. Fungsi ini mengambil nilai jam, menit, dan detik dari variabel currentTime, kemudian menampilkan setiap komponen waktu dengan menambahkan angka nol di depan jika nilainya kurang dari 10 agar format tampilan selalu dua digit. Fungsi *RTC_begin()* digunakan untuk menginisialisasi modul RTC. Fungsi ini memeriksa apakah modul RTC terdeteksi dengan benar. Jika modul tidak terdeteksi, maka akan menampilkan pesan error di serial monitor.

Selain itu, fungsi ini juga memeriksa apakah modul RTC kehilangan daya (misalnya karena baterai cadangan habis), dan jika iya, maka waktu RTC akan diatur ulang ke waktu kompilasi program menggunakan fungsi *rtc.adjust()*. Fungsi *RTC_update_time_and_get_temp()* berfungsi untuk memperbarui variabel waktu

currentTime dengan membaca waktu saat ini dari modul RTC. Fungsi ini juga mengembalikan nilai suhu yang diukur oleh sensor temperatur internal pada modul RTC. Dengan demikian, fungsi ini memungkinkan pembacaan waktu dan suhu secara bersamaan untuk aplikasi monitoring waktu dan lingkungan. Secara keseluruhan, ketiga fungsi ini berperan penting dalam pengelolaan waktu dan tampilan waktu pada sistem berbasis mikrokontroler ESP32 yang menggunakan modul RTC untuk menjaga akurasi waktu

j. Konfigurasi Arduino IOT Cloud



```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[]  = "c384e2bd-c8dc-4dff-bacd-4c25163a7f0b";

const char SSID[]               = SECRET_SSID;    // Network SSID (name)
const char PASS[]               = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[]         = SECRET_DEVICE_KEY;    // Secret device password

void onNH3Change();
void onPM25Change();
void onTEMPERATUREChange();
void onCO2Change();
void onHUMIDITYChange();

float nH3;
float pM25;
CloudTemperatureSensor tEMPERATURE;
int cO2;
CloudRelativeHumidity hUMIDITY;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(nH3, READWRITE, ON_CHANGE, onNH3Change);
  ArduinoCloud.addProperty(pM25, READWRITE, ON_CHANGE, onPM25Change);
  ArduinoCloud.addProperty(tEMPERATURE, READWRITE, ON_CHANGE, onTEMPERATUREChange);
  ArduinoCloud.addProperty(cO2, READWRITE, ON_CHANGE, onCO2Change);
  ArduinoCloud.addProperty(hUMIDITY, READWRITE, ON_CHANGE, onHUMIDITYChange);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

Gambar 4. 12 Konfigurasi Arduino IOT Cloud

Gambar 4.11 merupakan kode program bagian dari pengaturan koneksi dan properti variabel dalam platform Arduino IoT Cloud untuk proyek monitoring kualitas udara menggunakan mikrokontroler ESP32. Pada kode ini, terdapat definisi kredensial perangkat seperti DEVICE_LOGIN_NAME, SSID, PASS, dan DEVICE_KEY yang digunakan untuk menghubungkan perangkat ke jaringan WiFi dan Arduino IoT Cloud secara aman. Fungsi *initProperties()* berperan penting dalam menginisialisasi properti yang akan dikomunikasikan dengan cloud. Properti tersebut meliputi variabel gas amonia (nH3), partikel debu PM2.5, suhu, konsentrasi CO_2, dan kelembapan relatif. Setiap properti didaftarkan dengan mode baca-tulis (READWRITE), dan menggunakan trigger *ON_CHANGE* yang berarti fungsi call back seperti *onNH3Change()* akan dipanggil saat nilai properti tersebut

berubah. Call back ini memungkinkan perangkat merespons perubahan data secara real-time. Objek *WiFiConnectionHandler ArduinoIoTPreferredConnection* mengatur koneksi WiFi menggunakan SSID dan password yang telah ditentukan, sehingga perangkat dapat terhubung ke internet dan berkomunikasi dengan Arduino IoT Cloud. Secara keseluruhan, kode ini mengatur integrasi perangkat keras dengan layanan cloud, memungkinkan pengiriman dan penerimaan data sensor kualitas udara secara real-time, serta memudahkan pemantauan dan pengendalian perangkat dari jarak jauh melalui platform Arduino IoT Cloud.

k. Konfigurasi DMD P10

```
void P10_bottom(int val_temp, int val_humi, float val_dust) {
  display.drawFastHLine(0, 0, 64, 0xFF);  // Garis pembatas
  display.drawFastHLine(21, 0, 43, 0xFF);
  display.drawFastVLine(21, 0, 16, 0xFF);

  display.setFont(&TomThumb);
  display.setTextColor(0xFF);  // Set warna teks default untuk bagian ini

  display.setCursor(29, 15);
  display_jam();

  if (val_temp > temp_thres || val_humi > humi_thres) {
    P10_fill(0, 21, 8);
    P10_fill(0, 21, 16);
    display.setTextColor(0x00);
    display.setCursor(1, 7);
    display.print("T:");
    display.print(val_temp);
    display.setCursor(16, 7);
    display.print("C");

    display.setCursor(1, 15);
    display.print("H:");
    display.print(val_humi);
    display.setCursor(16, 15);
    display.print("%");
  }

  else {
    display.setTextColor(0xFF);
    display.setCursor(1, 7);
    display.print("T:");
    display.print(val_temp);
    display.setCursor(16, 7);
    display.print("C");

    display.setCursor(1, 15);
    display.print("H:");
    display.print(val_humi);
    display.setCursor(16, 15);
    display.print("%");
  }
```

Gambar 4. 13 Konfigurasi DMD P10

Gambar 4.12 merupakan kode konfigurasi DMD P10 untuk tampilan suhu, kelembapan dan konsentrasi PM 2.5. Fungsi *P10_bottom()* menampilkan suhu, kelembapan, dan waktu pada display dot matrix dengan garis pembatas untuk tata letak yang rapi. Jika suhu atau kelembapan melebihi ambang batas, area tertentu pada display disorot dengan latar gelap dan teks berwarna kontras untuk memberi peringatan. Jika masih dalam batas normal, data ditampilkan dengan warna teks terang. Suhu dan kelembapan ditampilkan dengan label dan satuan yang jelas, sedangkan waktu ditampilkan di posisi khusus, sehingga pengguna dapat memantau kondisi lingkungan secara real-time dengan mudah dan efektif.