BAB IV
PEMBUATAN ALAT

4.1 Pembuatan Prototipe

Dalam proses perancangan dan implementasi sistem peringatan dini kualitas udara berbasis ESP32, diperlukan suatu metode penelitian yang sistematis untuk memastikan setiap tahapan pengembangan dapat dilaksanakan secara terstruktur dan menghasilkan sistem yang berfungsi sesuai dengan tujuan penelitian. Bab ini menjelaskan tahapan perancangan dan pembuatan sistem yang meliputi proses perencanaan, perancangan, implementasi, serta pengujian untuk memastikan seluruh bagian sistem dapat bekerja secara optimal dan terintegrasi.

Setiap tahapan dalam pengembangan sistem memiliki keterkaitan yang erat sehingga diperlukan perencanaan yang matang agar perangkat yang dihasilkan mampu memenuhi kebutuhan pemantauan kualitas udara secara real-time. Secara umum, proses pembuatan sistem pada penelitian ini dibagi menjadi dua tahapan utama, yaitu sebagai berikut.

a. Perancangan Perangkat Keras (*Hardware*)

   Perangkat keras merupakan bagian fisik dari sistem yang berfungsi untuk mendukung proses akuisisi data, pemrosesan data, serta penyampaian informasi kepada pengguna. Tahap perancangan perangkat keras diawali dengan identifikasi kebutuhan sistem dan penentuan spesifikasi perangkat yang akan digunakan. Selanjutnya dilakukan proses perancangan rangkaian dan integrasi antarperangkat untuk membentuk suatu sistem yang sesuai dengan kebutuhan penelitian. Tahapan ini bertujuan untuk menghasilkan perangkat yang mampu beroperasi secara stabil, efisien, dan sesuai dengan rancangan yang telah ditetapkan.

b. Perancangan Perangkat Lunak (*Software*)

   Tahap selanjutnya adalah perancangan perangkat lunak yang berfungsi untuk mengendalikan jalannya sistem secara keseluruhan. Perangkat lunak bertugas mengatur proses pembacaan data, pengolahan data, pengambilan keputusan berdasarkan parameter yang telah ditentukan, serta menampilkan

hasil pengolahan data pada LCD I2C sebagai media informasi bagi pengguna.

Proses pengembangan perangkat lunak diawali dengan penyusunan algoritma dan perancangan alur kerja sistem, kemudian dilanjutkan dengan implementasi program menggunakan Arduino IDE. Setelah proses pemrograman selesai, dilakukan pengujian dan evaluasi untuk memastikan seluruh fungsi sistem dapat berjalan sesuai dengan kebutuhan dan spesifikasi yang telah direncanakan.

Langkah awal dalam penyelesaian penelitian ini adalah menyusun perencanaan yang matang dan merumuskan konsep sistem yang akan dikembangkan. Tahapan tersebut menjadi landasan dalam proses perancangan perangkat keras maupun perangkat lunak sehingga sistem yang dihasilkan mampu beroperasi secara optimal, terintegrasi, dan sesuai dengan tujuan penelitian yang telah ditetapkan.

4.2 Alat dan Bahan

Identifikasi kebutuhan merupakan tahapan awal yang dilakukan untuk menentukan seluruh sumber daya yang diperlukan dalam proses pengembangan sistem. Tahap ini bertujuan untuk memastikan bahwa perangkat keras dan perangkat lunak yang digunakan telah sesuai dengan kebutuhan fungsional serta spesifikasi sistem yang dirancang. Tabel 4.1 menunjukkan daftar kebutuhan yang digunakan sebagai acuan dalam proses perancangan, implementasi, dan pengujian sistem. Melalui identifikasi kebutuhan yang sistematis, proses pengembangan dapat dilakukan secara lebih efektif, terstruktur, dan mampu meminimalkan kesalahan dalam pemilihan komponen maupun pengalokasian sumber daya selama penelitian berlangsung.

Tabel 4- 1 Bahan Pembuatan Tugas Akhir

NO	KOMPONEN	SPESIFIKASI	JUMLAH
1	Mikrokontroler	Wroom	1
2	Sensor Gas NH3 dan CO_2	MQ-135	1
3	Sensor PM2.5	PMS5003	1
4	Suhu dan Kelembapan	DHT 22	1
5	Display	LCD I2C	1

NO	KOMPONEN	SPESIFIKASI	JUMLAH
6	Output Audio	Buzzer	1
7	Output Visual	LED WS2812	1
8	Modul Step-Up	MT3608	1
9	Modul Charger	TP4056	1
10	Suplai Daya	LP 403040	1
11	Resistor	10 kΩ	1
12	Resistor	330 Ω	1
13	Kapasitor	470 µF	2
14	Terminal Blok	Pitch 5.08 mm	1
15	Kabel	JST 3 Pin	3
16	Kabel	JST 4 Pin	2
17	Kabel	JST 5 Pin	1
18	PCB	Single Layer	1
19	Switch	ON\OFF	1

Tabel 4- 2 Alat Yang Digunakan

NO	ALAT	JUMLAH
1	Multimeter	1
2	Solder	1
3	Penyedot Timah	1
4	Tang Potong	1
5	Obeng (+)	1
6	Obeng (-)	1
7	Bor	1
8	Mata Bor	1

4.3 Pembuatan Perangkat Keras

Proses perancangan perangkat keras diawali dengan studi literatur yang dilakukan melalui pengumpulan berbagai referensi dari sumber ilmiah yang relevan, seperti buku, artikel ilmiah, jurnal, serta penelitian terdahulu yang berkaitan dengan sistem yang dikembangkan. Tahap ini bertujuan untuk

memperoleh pemahaman yang komprehensif mengenai konsep, metode, dan teknologi yang dapat diterapkan dalam penelitian.

Berdasarkan hasil studi literatur, dilakukan analisis kebutuhan sistem untuk menentukan spesifikasi perangkat keras yang diperlukan agar mampu mendukung fungsi dan kinerja sistem secara optimal. Tahap analisis ini mencakup identifikasi kebutuhan perangkat, karakteristik operasional, serta kesesuaian antar komponen yang akan digunakan dalam sistem.

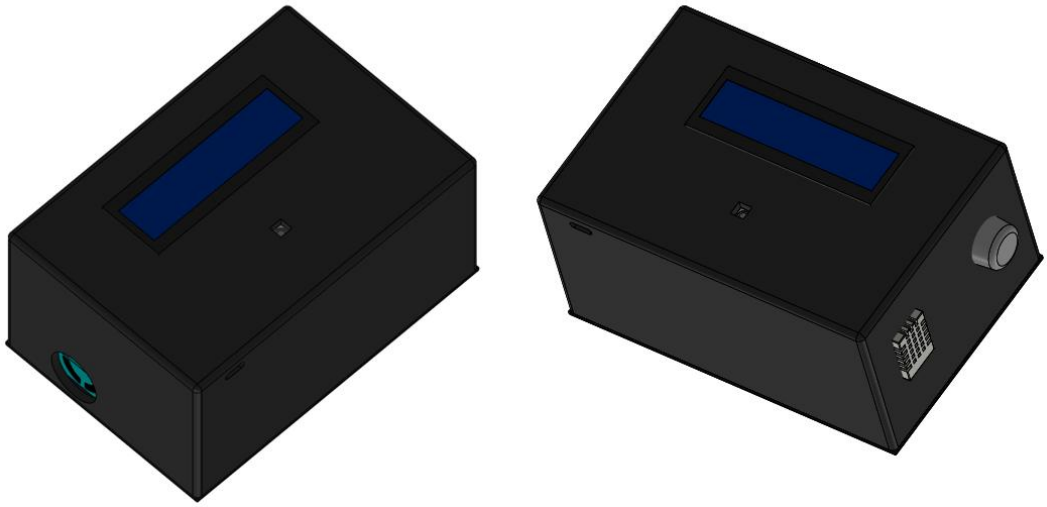
Selanjutnya, dilakukan proses perancangan rangkaian perangkat keras dengan mempertimbangkan integrasi dan konektivitas antar komponen agar dapat bekerja secara terkoordinasi. Perancangan dilakukan secara sistematis untuk memastikan setiap bagian sistem dapat menjalankan fungsinya sesuai dengan kebutuhan yang telah ditetapkan.

Setelah tahap perancangan selesai, dilakukan proses implementasi perangkat keras melalui perakitan seluruh komponen berdasarkan desain yang telah dibuat. Tahap ini dilanjutkan dengan pengujian konektivitas dan verifikasi fungsi perangkat untuk memastikan seluruh bagian sistem beroperasi dengan baik. Hasil pengujian digunakan sebagai dasar evaluasi guna memastikan bahwa perangkat yang dikembangkan telah memenuhi spesifikasi teknis dan kebutuhan fungsional yang ditetapkan dalam penelitian.

4.3.1 Perancangan Mekanik

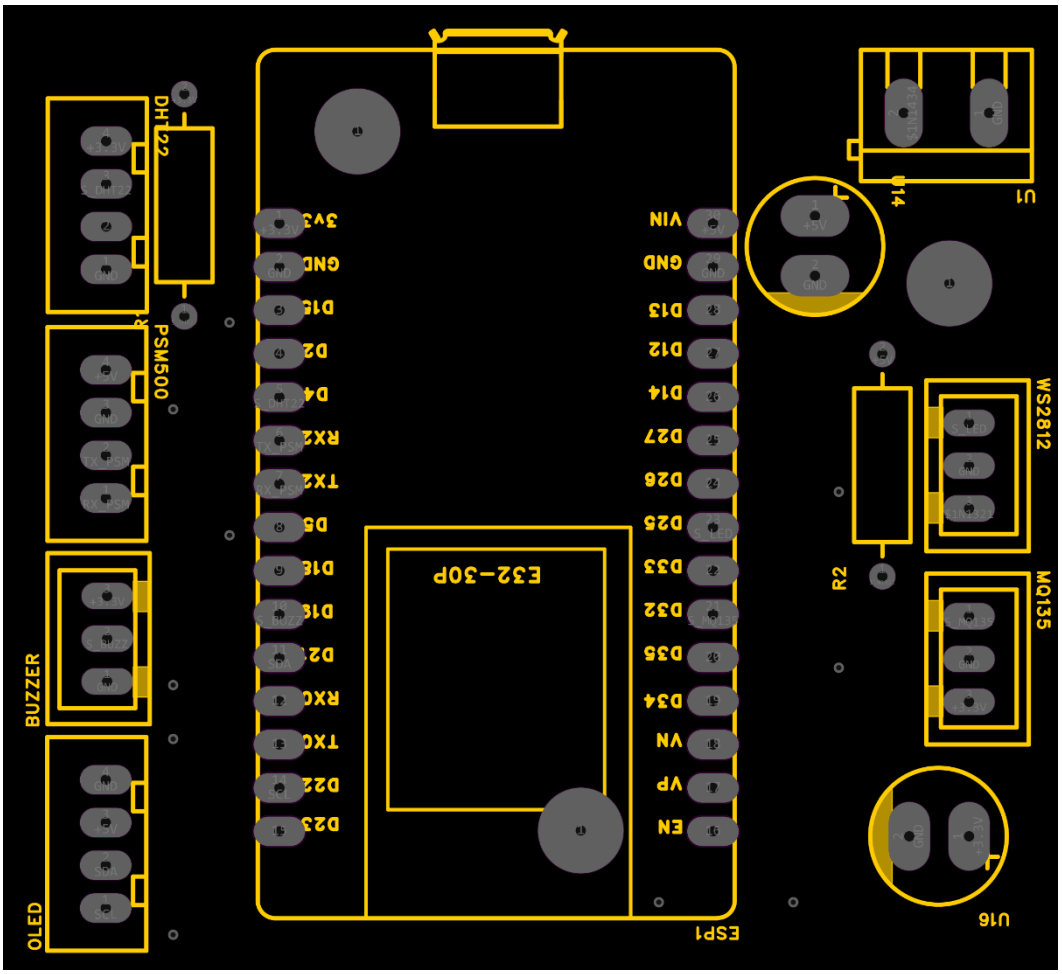
Setelah tahap perancangan dan perakitan perangkat keras selesai, langkah berikutnya adalah melakukan perancangan mekanik berupa desain casing alat. Perancangan mekanik bertujuan untuk menghasilkan struktur fisik yang mampu menempatkan seluruh komponen, seperti ESP32, sensor MQ-135, sensor PMS5003, sensor DHT22, LCD, buzzer, LED WS2812 secara rapi dan terintegrasi. Selain itu, casing dirancang untuk melindungi komponen elektronik dari benturan dan debu, memberikan ruang sirkulasi udara yang memadai agar sensor dapat melakukan pengukuran secara optimal, serta mendukung konsep portabel sehingga alat mudah dibawa dan digunakan di berbagai lokasi. Dengan perancangan mekanik yang baik, sistem pemantauan dan peringatan dini kualitas udara dapat beroperasi

secara optimal, memiliki keandalan yang tinggi, serta memberikan kemudahan dalam proses penggunaan maupun pemeliharaan.



Gambar 4. 1 Desain 3D Perancagan Mekanik

4.3.2 Perancagan Alat



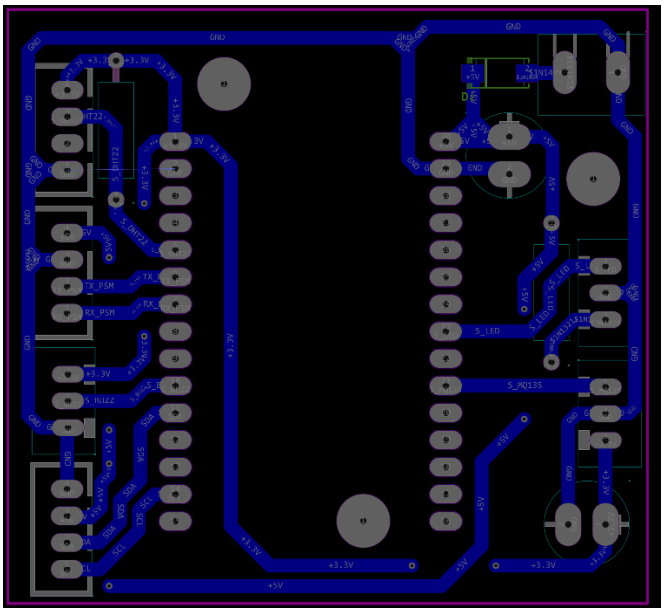
Gambar 4. 2 Posisi Penempatan Komponen

Setelah proses perakitan perangkat keras selesai dilakukan, tahapan berikutnya adalah perancangan alat secara menyeluruh yang meliputi integrasi seluruh komponen elektronik ke dalam suatu sistem yang memiliki fungsi dan struktur yang sesuai dengan tujuan penelitian. Tahap perancangan ini bertujuan untuk memastikan bahwa perangkat yang telah dikembangkan mampu beroperasi secara optimal sesuai dengan kondisi penggunaannya, serta memenuhi aspek fungsional, kemudahan pengoperasian, portabilitas, dan keandalan sistem secara keseluruhan.

Pada tahap ini, penempatan setiap komponen dirancang dengan mempertimbangkan efisiensi penggunaan ruang, kemudahan proses perakitan, aksesibilitas saat pemeliharaan, serta kestabilan kinerja perangkat. Tata letak komponen tersebut selanjutnya dijadikan sebagai acuan dalam proses perancangan

dan pencetakan Printed Circuit Board (PCB) agar koneksi antar komponen dapat tersusun secara sistematis dan sesuai dengan desain yang telah ditetapkan. Gambar 4.2 menunjukkan posisi masing-masing komponen yang digunakan sebagai dasar dalam proses pembuatan PCB pada Sistem Peringatan Dini Kualitas Udara Portabel berbasis ESP32.

4.3.3 Perancangan PCB



Gambar 4. 3 Perancangan PCB

Setelah proses pembuatan perangkat keras dan perancangan alat selesai dilakukan, tahapan selanjutnya adalah melakukan perancangan Printed Circuit Board (PCB) sebagai media integrasi seluruh komponen elektronik yang digunakan dalam sistem. Perancangan PCB bertujuan untuk menyusun hubungan antar komponen secara lebih terstruktur, sehingga menghasilkan sistem yang lebih ringkas, stabil, dan efisien dibandingkan dengan penggunaan media perakitan sementara, seperti breadboard.

Perancangan PCB juga dilakukan untuk memastikan bahwa setiap jalur koneksi listrik antar komponen tersusun sesuai dengan kebutuhan sistem, sehingga dapat meminimalkan terjadinya kesalahan koneksi serta meningkatkan keandalan perangkat dalam jangka panjang. Selain itu, tata letak komponen pada PCB dirancang dengan mempertimbangkan aspek efisiensi ruang, kemudahan proses

perakitan, kemudahan pemeliharaan, serta mendukung karakteristik portabel dari alat yang dikembangkan.

Melalui perancangan PCB yang baik, Sistem Peringatan Dini Kualitas Udara Portabel berbasis ESP32 dapat diwujudkan dalam bentuk produk akhir yang lebih kompak, memiliki kualitas koneksi listrik yang optimal, serta lebih mudah untuk direalisasikan dan direproduksi apabila diperlukan pengembangan lebih lanjut. Gambar 4.3 menunjukkan rancangan PCB yang akan digunakan sebagai acuan dalam proses pencetakan dan perakitan akhir perangkat.

4.4 Pembuatan Perangkat Lunak

Pada tahap ini dilakukan pengembangan perangkat lunak melalui proses perancangan, implementasi, dan penulisan kode program menggunakan Arduino IDE. Pengembangan perangkat lunak ini bertujuan untuk mengimplementasikan algoritma sistem pada mikrokontroler ESP32 sesuai dengan rancangan yang telah dibuat. Tahap ini memastikan bahwa sistem dapat beroperasi sesuai dengan kebutuhan dan spesifikasi yang telah ditentukan. Dengan demikian, perangkat lunak yang dikembangkan mampu mengendalikan perangkat keras secara optimal, melakukan proses pengukuran dan pengolahan data, serta menjalankan komunikasi data secara efektif sesuai dengan kebutuhan aplikasi.

4.4.1 Kalibrasi Sensor MQ135 untuk Pengukuran CO_2

Kalibrasi sensor MQ135 bertujuan untuk menentukan nilai resistansi acuan (Ro), yaitu resistansi sensor pada kondisi udara bersih yang konsentrasi CO_2-nya telah diketahui. Nilai Ro bersifat khas untuk setiap unit sensor sehingga tidak dapat diambil langsung dari datasheet, melainkan harus ditentukan secara empiris. Nilai Ro inilah yang menjadi pembanding bagi resistansi sensor saat pengukuran (Rs), sehingga rasio Rs/Ro dapat dikonversi menjadi konsentrasi CO_2 dalam satuan ppm.

Kalibrasi dilakukan pada kondisi operasi yang sama persis dengan kondisi pengukuran sebenarnya, yaitu sensor ditenagai oleh baterai LiPo melalui modul penaik tegangan (MT3608) yang telah disetel pada 5 V, dengan keluaran analog sensor (AOUT) dihubungkan ke pin ADC ESP32 melalui rangkaian pembagi tegangan 1 kΩ : 1 kΩ. Kesamaan kondisi ini penting agar nilai Ro yang diperoleh

tetap konsisten ketika sistem dioperasikan. Adapun langkah-langkah kalibrasi yang dilakukan adalah sebagai berikut:

1. Sensor dirangkai pada kondisi operasi penuh (suplai 5 V, pembagi tegangan terpasang, ditenagai baterai LiPo).
2. Sensor dipanaskan (warm-up) selama kurang lebih 20–30 menit hingga pembacaan resistansinya stabil.
3. Sensor ditempatkan pada udara terbuka/bersih dengan konsentrasi CO_2 acuan sebesar ± 415 ppm.
4. Program kalibrasi dijalankan untuk membaca tegangan keluaran sensor, menghitung Rs, dan merata-ratakan sejumlah pembacaan guna mengurangi derau.
5. Nilai Ro dihitung secara otomatis dan ditampilkan, kemudian dicatat setelah pembacaannya stabil.
6. Nilai Ro hasil kalibrasi dimasukkan ke program utama untuk digunakan pada proses pengukuran.

Perhitungan nilai Ro pada program kalibrasi mengacu pada penurunan persamaan karakteristik sensor, yaitu Ro = Rs / (Cacuan/a)1/b, dengan Cacuan adalah konsentrasi CO_2 acuan. Pembacaan resistansi sensor Rs diperoleh dari tegangan keluaran sensor menggunakan fungsi analogReadMilliVolts() agar pembacaan ADC sesuai dengan kalibrasi pabrik ESP32. Potongan program kalibrasi yang digunakan ditunjukkan pada Gambar 4.4.

```
// ---- Konfigurasi ----
const int   MQ135Pin     = 32;
const float RLOAD        = 10000.0;   // resistor beban (ohm)
const float VCC          = 5.0;       // tegangan suplai sens
const float DIVIDER_GAIN = 2.0;       // pembagi 1k : 1k
const float a = 110.7432567;          // koefisien kurva CO2
const float b = -2.856935538;
const float PPM_UDARA_BERSIH = 415.0; // konsentrasi acuan

float targetRatio;

// ---- Membaca resistansi sensor (Rs) ----
float bacaRs() {
  float vPin = analogReadMilliVolts(MQ135Pin) / 1000.0;
  float vOut = vPin * DIVIDER_GAIN;
  if (vOut < 0.01) vOut = 0.01;
  if (vOut > VCC)  vOut = VCC - 0.001;
  return RLOAD * (VCC - vOut) / vOut;
}

void setup() {
  Serial.begin(9600);
  analogReadResolution(12);
  analogSetPinAttenuation(MQ135Pin, ADC_11db);
  targetRatio = pow(PPM_UDARA_BERSIH / a, 1.0 / b);
}

void loop() {
  // rata-rata 50 pembacaan Rs
  float rS = 0;
  for (int i = 0; i < 50; i++) { rS += bacaRs(); delay(20); }
  rS /= 50.0;

  // hitung Ro dari kondisi udara bersih
  float Ro = rS / targetRatio;

  Serial.print("Rs = ");  Serial.print(rS, 0);
  Serial.print(" ohm | Ro = "); Serial.println(Ro, 0);
  delay(1000);
}
```

Gambar 4. 4 Program Kalibrasi Sensor MQ135 untuk Pengukuran CO_2

Berdasarkan proses kalibrasi yang dilakukan pada udara bersih dengan konsentrasi acuan 415 ppm, diperoleh nilai resistansi sensor Rs sebesar 82.251 Ω (± 82,25 kΩ). Dengan menggunakan persamaan kalibrasi, diperoleh nilai resistansi acuan Ro sebesar 131.731 Ω (± 131,73 kΩ). Rasio Rs/Ro pada kondisi kalibrasi bernilai sekitar 0,62, yang apabila disubstitusikan kembali ke dalam persamaan konversi menghasilkan pembacaan ± 415 ppm. Hal ini menunjukkan bahwa nilai Ro hasil kalibrasi telah sesuai dengan konsentrasi acuan. Rangkuman parameter dan hasil kalibrasi ditunjukkan pada Tabel 4-3.

Tabel 4- 3 Parameter dan hasil kalibrasi sensor MQ135

Parameter	Nilai
Tegangan suplai sensor (Vcc)	5 V
Resistor beban (RL)	10 kΩ
Pembagi tegangan AOUT	1 kΩ : 1 kΩ (penguatan 2,0)
Konsentrasi CO_2 acuan	415 ppm
Resistansi sensor udara bersih (Rs)	82.251 Ω

Parameter	Nilai
Resistansi acuan hasil kalibrasi (Ro)	131.731 Ω
Rasio Rs/Ro	0,62
Hasil pembacaan verifikasi	± 415 ppm

Nilai Ro yang diperoleh selanjutnya digunakan sebagai konstanta tetap pada program utama. Perlu dicatat bahwa kalibrasi ini dilakukan pada satu titik acuan (single-point calibration) menggunakan konsentrasi CO_2 udara luar, sehingga akurasi pembacaan paling baik berada di sekitar rentang konsentrasi acuan tersebut. Mengingat MQ135 bukan merupakan sensor CO_2 spesifik dan bersifat sensitif terhadap beberapa gas sekaligus, hasil pembacaan lebih tepat diinterpretasikan sebagai estimasi kecenderungan kadar CO_2 dan bukan sebagai pengukuran absolut.

4.4.2 Konfigurasi MQ135 untuk Pengukuran CO_2

```
// KONFIGURASI MQ135 UNTUK PENGUKURAN CO2
#define MQ135_PIN 32
const float a = 110.7432567;
const float b = -2.856935538;
const double ppmCalibration = 415.0;
float RO = 503994.0;
const float RLOAD = 10000.0;
float minPPM = 0;
float maxPPM = 0;

void MQ135Begin() {
  pinMode(MQ135_PIN, INPUT);
  analogSetPinAttenuation(MQ135_PIN, ADC_11db);
}

void findMaxMinPPM() {
  minPPM = pow((1000 / a), (1 / b));
  maxPPM = pow((10 / a), (1 / b));
}

float getPPM() {
  float adcRaw = analogRead(MQ135_PIN);
  float RsValue = ((4095.0 * RLOAD) / adcRaw) - RLOAD;
  float rSrO = RsValue / RO;
  if (rSrO < maxPPM && rSrO > minPPM) {
    float ppmValue = a * pow((float)RsValue / (float)RO, b);
    AverageValuePush(ppmValue);
    float ppmValue_avg = AverageValue();
    return ppmValue_avg;
  } else return -1.0;
}

int warnScaleMQ(float PPM) {
  if (PPM < PPM_BAIK) return 1;
  else if (PPM < PPM_RINGAN) return 2;
  else if (PPM < PPM_BAHAYA) return 3;
  else return 4;
}
```

Gambar 4. 5 Konfigurasi MQ135 untuk Pengukuran CO_2

Gambar 4.5 merupakan kode program yang mengimplementasikan konfigurasi sensor MQ135 yang digunakan untuk mengukur konsentrasi gas CO_2 di udara. Fungsi MQ135Begin() bertugas menginisialisasi pin analog yang terhubung dengan sensor MQ135 serta mengatur atenuasi ADC menggunakan ADC_11db agar pembacaan tegangan lebih presisi pada rentang penuh. Fungsi findMaxMinPPM() menghitung batas nilai rasio Rs/Ro minimum dan maksimum berdasarkan konstanta kalibrasi a dan b, yang digunakan sebagai batas validitas data sebelum nilai PPM dihitung. Fungsi getPPM() membaca nilai ADC dari sensor, menghitung nilai resistansi sensor (Rs), kemudian mengonversinya menjadi nilai konsentrasi CO_2 dalam satuan PPM menggunakan persamaan kalibrasi. Apabila rasio Rs/Ro berada di luar rentang minPPM dan maxPPM, fungsi akan mengembalikan nilai -1 sebagai indikator error. Untuk menjaga kestabilan pembacaan, nilai PPM dirata-rata menggunakan buffer melalui fungsi AverageValuePush() dan AverageValue() sebelum ditampilkan. Selain itu, fungsi warnScaleMQ() digunakan untuk mengklasifikasikan tingkat kualitas udara berdasarkan nilai PPM ke dalam kategori Baik, Ringan, Sedang, dan Bahaya sesuai ambang batas yang telah ditentukan. Dengan konfigurasi ini, sistem mampu melakukan pengukuran konsentrasi CO_2 secara akurat dan real-time sebagai salah satu parameter utama dalam pemantauan kualitas udara berbasis mikrokontroler ESP32.

4.4.3 Konfigurasi PMS5003

```ino
// KONFIGURASI PMS5003
#define PMS5003_RX  17
#define PMS5003_TX  16

struct pms5003data {
  uint16_t framelen;
  uint16_t pm10_standard, pm25_standard, pm100_standard;
  uint16_t pm10_env, pm25_env, pm100_env;
  uint16_t particles_03um, particles_05um, particles_10um,
           particles_25um, particles_50um, particles_100um;
  uint16_t unused;
  uint16_t checksum;
};

HardwareSerial pm25Serial(2);
pms5003data PMS25;

void PMBegin() {
  Serial.println("INITIALIZATION PM2.5 SENSOR");
  pm25Serial.begin(9600, SERIAL_8N1, PMS5003_TX, PMS5003_RX);
}

bool PMStreamData(Stream *s) {
  if (!s->available()) return false;
  if (s->peek() != 0x42) { s->read(); return false; }
  if (s->available() < 32) return false;

  uint8_t buffer[32];
  uint16_t sum = 0;
  s->readBytes(buffer, 32);
  for (uint8_t i = 0; i < 30; i++) sum += buffer[i];

  uint16_t buffer_u16[15];
  for (uint8_t i = 0; i < 15; i++) {
    buffer_u16[i] = buffer[2 + i * 2 + 1];
    buffer_u16[i] += (buffer[2 + i * 2] << 8);
  }

  memcpy((void *)&PMS25, (void *)buffer_u16, 30);

  if (sum != PMS25.checksum) {
    Serial.println("Checksum failure");
    return false;
  }
  return true;
}

void PMGetData() {
  if (PMStreamData(&pm25Serial)) {
    Serial.print("PM 2.5: ");
    Serial.println(PMS25.pm25_env);
    PM25 = PMS25.pm25_env - calibrationValue;
  }
}

int warnScalePM(float PM25) {
  if (PM25 < PM25_BAIK) return 1;
  else if (PM25 < PM25_RINGAN) return 2;
  else if (PM25 < PM25_BAHAYA) return 3;
  else return 4;
}
```

Gambar 4. 6 Konfigurasi PMS5003

Gambar 4.6 merupakan kode program yang mengimplementasikan konfigurasi sensor PMS5003 yang digunakan untuk mengukur konsentrasi partikulat PM2.5 di udara. Fungsi PMBegin() bertugas menginisialisasi komunikasi serial UART2 pada ESP32 dengan baud rate 9600 yang terhubung pada pin TX dan RX sensor PMS5003. Struktur data pms5003data digunakan untuk menampung hasil parsing data biner yang dikirimkan sensor, meliputi nilai standar dan nilai lingkungan dari PM1.0, PM2.5, dan PM10, jumlah partikel pada berbagai ukuran, serta nilai

checksum. Fungsi PMStreamData() memvalidasi byte awal frame data (0x42), memastikan jumlah data yang tersedia mencukupi 32 byte, kemudian menyusun ulang byte tersebut menjadi struktur data 16-bit dan memverifikasi keabsahan data melalui perhitungan checksum. Apabila checksum tidak sesuai, fungsi akan mengembalikan nilai false yang menandakan data tidak valid. Fungsi PMGetData() memanggil PMStreamData() untuk memperoleh data terbaru dan menyimpan nilai PM2.5 lingkungan setelah dikurangi nilai kalibrasi. Selanjutnya, fungsi warnScalePM() digunakan untuk mengklasifikasikan tingkat kualitas udara berdasarkan nilai PM2.5 ke dalam kategori Baik, Ringan, Sedang, dan Bahaya. Dengan konfigurasi ini, sistem dapat memperoleh data partikulat udara secara berkala dan akurat sebagai parameter tambahan dalam sistem pemantauan kualitas udara.

4.4.4 Konfigurasi DHT22

```
// KONFIGURASI DHT22
#define DHT_PIN 4
#define DHTTYPE DHT22

DHT dht(DHT_PIN, DHTTYPE);
double Temperature, Humidity;

void DHTBegin() {
  Serial.println("INITIALIZATION DHT22 SENSOR");
  dht.begin();
}

void DHTReadData() {
  Temperature = dht.readTemperature();
  Humidity = dht.readHumidity();
  Serial.print("TEMPERATURE : ");
  Serial.println(Temperature, 2);
  Serial.print("HUMIDITY : ");
  Serial.println(Humidity, 2);
}
```

Gambar 4. 7 Konfigurasi DHT22

Gambar 4.7 merupakan kode program yang mengimplementasikan konfigurasi sensor DHT22 yang digunakan untuk mengukur suhu dan kelembaban udara. Fungsi DHTBegin() bertugas menginisialisasi objek dht yang terhubung pada pin DHT_PIN dengan tipe sensor DHTTYPE (DHT22) menggunakan pustaka DHT. Fungsi DHTReadData() melakukan pembacaan data suhu melalui dht.readTemperature() dan data kelembaban melalui dht.readHumidity(), kemudian menyimpan hasilnya ke dalam variabel global Temperature dan Humidity. Kedua nilai tersebut selanjutnya ditampilkan melalui Serial Monitor untuk keperluan pemantauan dan debugging. Data suhu dan kelembaban yang diperoleh dari

konfigurasi ini digunakan sebagai salah satu parameter pendukung dalam sistem pemantauan kualitas udara berbasis mikrokontroler ESP32.

4.4.5 Konfigurasi LCD 16X2 I2C

```
// KONFIGURASI LCD 16X2 I2C
#define LCD_ADDRESS 0x27
#define LCD_COLUMNS 16
#define LCD_ROWS 2

LiquidCrystal_I2C lcd(0x27, 16, 2);

void lcdBegin() {
  lcd.init();
  lcd.backlight();
}

void displayCentered(String text, int row) {
  int len = text.length();
  int spaces = (LCD_COLUMNS - len) / 2;
  String padding = "";
  for (int i = 0; i < spaces; i++) {
    padding += " ";
  }
  lcd.setCursor(0, row);
  lcd.print(padding + text);
}
```

Gambar 4. 8 Konfigurasi LCD16X2 I2C

Gambar 4.8 merupakan kode program yang mengimplementasikan konfigurasi modul LCD 16x2 berbasis I2C yang digunakan sebagai media penampil hasil pembacaan sensor. Objek lcd dideklarasikan menggunakan pustaka LiquidCrystal_I2C dengan alamat I2C 0x27 serta ukuran 16 kolom dan 2 baris. Fungsi lcdBegin() bertugas menginisialisasi komunikasi I2C pada LCD melalui lcd.init() dan mengaktifkan lampu latar (backlight) agar tampilan dapat terbaca dengan jelas. Fungsi displayCentered() digunakan untuk menampilkan teks pada baris tertentu dengan posisi rata tengah, yaitu dengan menghitung panjang karakter teks kemudian menambahkan spasi pada sisi kiri sesuai jumlah kolom LCD yang tersedia. Konfigurasi ini memungkinkan sistem menampilkan informasi seperti status warmup sensor maupun data hasil pengukuran secara rapi dan mudah dibaca oleh pengguna.

4.4.6 Konfigurasi Buzzer

```
// KONFIGURASI BUZZER
#define BUZZER_PIN 19

void buzzerBegin() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH);
}

void updateBuzzer() {
  int pmStatus  = warnScalePM(PM25);
  int ppmStatus = WarmUp ? 1 : warnScaleMQ(CO2_PPM);

  if (pmStatus == 4 || ppmStatus == 4) digitalWrite(BUZZER_PIN, LOW);
  else digitalWrite(BUZZER_PIN, HIGH);
}
```

Gambar 4. 9 Konfigurasi Buzzer

Gambar 4.9 merupakan kode program yang mengimplementasikan konfigurasi buzzer sebagai indikator peringatan dini terhadap kondisi udara yang berbahaya. Fungsi buzzerBegin() bertugas mengatur pin BUZZER_PIN sebagai output serta memberikan kondisi awal HIGH agar buzzer dalam keadaan tidak aktif, mengingat modul buzzer yang digunakan bersifat active-low. Fungsi updateBuzzer() memeriksa status kualitas udara berdasarkan nilai warnScalePM() dan warnScaleMQ(); apabila salah satu status menunjukkan kategori Bahaya (nilai 4), maka buzzer akan diaktifkan dengan memberikan logika LOW pada pin buzzer, sedangkan pada kondisi lainnya buzzer akan dinonaktifkan dengan logika HIGH. Dengan konfigurasi ini, buzzer berfungsi sebagai peringatan audio secara real-time ketika tingkat pencemaran udara mencapai kategori berbahaya.

4.4.7 Konfigurasi LED

```
// KONFIGURASI LED
#define NUM_LEDS 1
#define DATA_PIN 25
CRGB leds[NUM_LEDS];

void FastLEDBegin() {
  FastLED.addLeds<WS2812, DATA_PIN, GRB>(leds, NUM_LEDS);
  FastLED.setBrightness(80);
  leds[0] = CRGB::Black;
  FastLED.show();
}

void updateLEDStatus() {
  int pmStatus = warnScalePM(PM25);
  int ppmStatus = WarmUp ? 1 : warnScaleMQ(CO2_PPM);
  int status = max(pmStatus, ppmStatus);

  switch (status) {
    case 1: leds[0] = CRGB::Green;  break;  // Baik
    case 2: leds[0] = CRGB::Yellow; break;  // Ringan
    case 3: leds[0] = CRGB::Orange; break;  // Sedang
    case 4: leds[0] = CRGB::Red;    break;  // Bahaya
    default: leds[0] = CRGB::Blue;  break;  // Error
  }
  FastLED.show();
}
```

Gambar 4. 10 Konfigurasi LED

Gambar 4.10 merupakan kode program yang mengimplementasikan konfigurasi LED RGB berbasis WS2812 menggunakan pustaka FastLED sebagai indikator visual status kualitas udara. Fungsi FastLEDBegin() bertugas menginisialisasi LED yang terhubung pada DATA_PIN dengan tipe WS2812 dan urutan warna GRB, mengatur tingkat kecerahan (brightness) sebesar 80, serta mengatur kondisi awal LED dalam keadaan mati (Black). Fungsi updateLEDStatus() menentukan status tertinggi antara kondisi PM2.5 (warnScalePM()) dan kondisi CO_2 (warnScaleMQ()) menggunakan fungsi max(), kemudian mengubah warna LED sesuai kategori, yaitu hijau untuk kondisi Baik, kuning untuk Ringan, oranye untuk Sedang, dan merah untuk Bahaya, sedangkan warna biru digunakan sebagai indikator kondisi error. Dengan konfigurasi ini, pengguna dapat memperoleh gambaran cepat mengenai status kualitas udara hanya dengan melihat warna LED tanpa perlu membaca layar LCD secara langsung.