



sistem tersemat

arduino, sensor dan aplikasi praktis

Zainal Akbar

Penerbit Al Arsy Media

SISTEM TERSEMAT

ARDUINO, SENSOR DAN APLIKASI PRAKTIS

PENULIS
ZAINAL AKBAR

EDITOR
NINIS INSIYAH MASYHUR

PENERBIT AL ARSY MEDIA

SISTEM TERSEMAT

ARDUINO, SENSOR DAN APLIKASI PRAKTIS

Penulis :

Zainal Akbar

Editor :

Ninis Insiyah Masyhur

Desain Cover :

Muh. Ardiansyah Latif

Tata Letak :

Muh. Ardiansyah Latif

Proofreader :

Maya Itasari

Ukuran :

Jml Hal Judul 2, Jml Hal isi naskah 77, Uk 148x210 mm

ISBN :

XXX XXX XX XXX X

Cetakan Pertama :

Oktober 2026

Hak Cipta 2026, Pada Penulis

Isi diluar tanggung jawab percetakan

Copyright © 2026 by Al Arsy Media

All Right Reserved

Hak cipta dilindungi undang-undang

Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa izin tertulis dari Penerbit.

PENERBIT AL ARSY MEDIA

Anggota IKAPI 072/SSL/2024

Jl. AMD, Perumahan Grand Sulawesi Antang E77,
Kel. Manggala, Kec. Manggala, Kota Makassar, Prov. Sul-Sel

Website: www.alarsymedia.com

E-mail: alarsymedia@gmail.com

KATA PENGANTAR

Puji syukur kehadiran Tuhan Yang Maha Esa atas terselesainya buku ini yang berjudul “Sistem Embedded : Arduino, Sensor, dan Aplikasi Praktis”. Buku ini disusun untuk memberikan pemahaman mendalam tentang konsep dan penerapan sistem embedded, yang saat ini memiliki peran penting dalam berbagai sektor, mulai dari otomasi hingga perangkat pintar sehari-hari.

Mikrokontroler menjadi pusat dari sistem embedded, memungkinkan integrasi antara perangkat keras dan lunak untuk menghasilkan sistem yang efisien dan responsif. Di sisi lain, sensor memainkan peran vital dalam menyediakan data real-time yang diperlukan untuk membuat keputusan yang akurat. Dengan perkembangan teknologi yang pesat, penguasaan kedua komponen ini menjadi semakin penting.

Melalui buku ini, pembaca akan dibimbing dari dasar teori hingga praktik, dengan contoh nyata yang diharapkan dapat diaplikasikan langsung. Diagram rangkaian, skema, dan kode program juga disertakan untuk mempermudah pemahaman. Buku ini ditujukan bagi mahasiswa, dosen, peneliti, dan praktisi yang ingin mendalami atau mengembangkan sistem berbasis mikrokontroler dan sensor.

Saya menyadari bahwa buku ini masih memiliki kekurangan, oleh karena itu, kritik dan saran yang membangun sangat saya harapkan. Semoga buku ini bermanfaat bagi semua pembaca. Terima kasih.

Selamat membaca dan terus berkarya

Makassar, Agustus 2026

Zainal Akbar

DAFTAR ISI

KATA PENGANTAR	IV
DAFTAR ISI	V
DAFTAR GAMBAR	VII
BAB 1 PENGANTAR MIKROKONTROLER	1
1.1 APA ITU MIKROKONTROLER?	1
1.2 GAMBARAN UMUM KELUARGA MIKROKONTROLER	1
1.3 APLIKASI MIKROKONTROLER DI DUNIA NYATA	3
BAB 2 ARSITEKTUR MIKROKONTROLER	5
2.1 KOMPONEN UTAMA: CPU, MEMORI, PORT I/O	5
2.2 MEMAHAMI PIN DAN PERIFERAL MIKROKONTROLER	11
2.3 MANAJEMEN DAYA DAN SISTEM CLOCK	12
BAB 3 PEMROGRAMAN MIKROKONTROLER	14
3.1 ARDUINO	14
3.2 BAHASA PEMROGRAMAN	17
3.3 LINGKUNGAN PENGEMBANGAN	17
3.4 MENULIS DAN MENGUNGGAH PROGRAM DASAR	19
3.5 MENGUNGGAH PROGRAM DASAR	21
BAB 4 INTERFACING DENGAN SENSOR	23
4.1 SENSOR DAN PRINSIP KERJA	23
4.2 JENIS SENSOR	24
4.3 MEMBACA SINYAL ANALOG DAN DIGITAL	28
4.4 CONTOH PRAKTIS: MENGHUBUNGKAN DAN MEMBACA DATA DARI SENSOR DHT22	32

<u>BAB 5 INTERFACING PERANGKAT KERAS</u>	<u>37</u>
5.1 KOMUNIKASI DENGAN PERANGKAT EKSTERNAL.....	38
5.2 MENGHUBUNGKAN DISPLAY DAN AKTUATOR	41
<u>BAB 6 DESAIN PERANGKAT LUNAK TERSEMAT (EMBEDEED SOFTWARE)</u>	<u>45</u>
6.1 MENULIS KODE YANG EFISIEN UNTUK MIKROKONTROLER.....	45
6.2 <i>INTERUPSI, TIMER</i> DAN KONTROL <i>REAL-TIME</i>	53
6.3 DEBUGGING DAN OPTIMASI KINERJA	59
<u>DAFTAR PUSTAKA</u>	<u>68</u>

DAFTAR GAMBAR

GAMBAR 1. MIKROKONTROLER 8-BIT	2
GAMBAR 2. MIKROKONTROLER 16-BIT	2
GAMBAR 3. MIKROKONTROLER 32-BIT	3
GAMBAR 4. PINOUT DIAGRAM ARDUINO UNO.....	15
GAMBAR 5. PINOUT DIAGRAM ARDUINO MEGA.....	17
GAMBAR 6. ARDUINO IDE 2	18
GAMBAR 7. MENU ARDUINO IDE 2.....	18
GAMBAR 8. MEMILIH <i>BOARD</i> YANG SESUAI	21
GAMBAR 9. TOMBOL VERIFIKASI DAN UNGGAH	22
GAMBAR 10. KOMPILASI BERHASIL	22
GAMBAR 11. SENSOR SUHU	24
GAMBAR 12. SENSOR MQ-7	25
GAMBAR 13. SENSOR PIR	26
GAMBAR 14. SENSOR ACCELEROMETER MPU6050.....	26
GAMBAR 15. SENSOR <i>LIGHT DEPENDENT RESISTOR (LDR)</i>	27
GAMBAR 16. <i>FLOW SENSOR</i>	27
GAMBAR 17. <i>FINGERPRINT</i>	27
GAMBAR 18. BENTUK SINYAL ANALOG DAN DIGITAL	30
GAMBAR 19. RANGKAIAN PEMBEACAN SENSOR ANALOG	31
GAMBAR 20. CONTOH PRAKTIS SENSOR LM35	32
GAMBAR 21. WIRING LCD I2C.....	42
GAMBAR 22. WIRING SERVO	42
GAMBAR 23. ALUR SISTEM BERBASIS EVENT DAN KONTROL.....	58
GAMBAR 24. TAHAP DEBUGGING.....	62

BAB 1

PENGANTAR MIKROKONTROLER

Bab ini menjelaskan tentang :

- Apa itu mikrokontroler?
- Gambaran umum keluarga mikrokontroler
- Aplikasi mikrokontroler di dunia nyata

1.1 Apa itu mikrokontroler?

Mikrokontroler adalah sebuah sirkuit terpadu (IC) yang dirancang untuk menjalankan tugas-tugas spesifik dalam sistem tertanam. Mikrokontroler menggabungkan CPU (Central Processing Unit), memori, dan periferal input/output (I/O) dalam satu chip, sehingga mampu melakukan berbagai fungsi kendali otomatis. Perangkat ini berfungsi sebagai "otak" dari sistem elektronik, membaca input dari sensor atau perangkat eksternal dan mengeluarkan perintah untuk menjalankan fungsi tertentu seperti mengendalikan motor, menyalakan lampu, atau membaca data dari sensor.

1.2 Gambaran umum keluarga mikrokontroler

Ada berbagai keluarga mikrokontroler yang tersedia, masing-masing memiliki fitur dan spesifikasi unik yang membuatnya ideal untuk aplikasi tertentu. Berikut adalah gambaran umum beberapa keluarga mikrokontroler yang populer:

1.2.1 Mikrokontroler 8-bit

Mikrokontroler 8-bit adalah yang paling sederhana dan biasanya digunakan dalam aplikasi yang tidak memerlukan pemrosesan data yang kompleks. CPU-nya dapat memproses data dalam ukuran 8-bit. Contoh mikrokontroler 8-bit adalah Mikrokontroler ARM Cortex ATmega328 yang digunakan dalam Arduino Uno dan PIC16 dari Microchip. Keluarga ini sering digunakan dalam aplikasi seperti kontrol perangkat rumah tangga, remote control, dan proyek DIY (*Do It Yourself*).



Gambar 1. Mikrokontroler 8-bit
(Sumber : www.mouser.co.id)

1.2.2 Mikrokontroler 16-bit

Mikrokontroler 16-bit menawarkan peningkatan performa dibandingkan mikrokontroler 8-bit. Mereka memiliki lebih banyak kapasitas untuk memproses data dan lebih cocok untuk aplikasi dengan kebutuhan komputasi yang lebih tinggi. Contoh dari mikrokontroler 16-bit adalah MSP430 dari Texas Instruments, yang sering digunakan dalam aplikasi yang memerlukan konsumsi daya rendah, seperti perangkat yang dioperasikan dengan baterai.



Gambar 2. Mikrokontroler 16-bit
(Sumber : www.pngegg.com)

1.2.3 Mikrokontroler 32-bit

Mikrokontroler 32-bit menawarkan kemampuan pemrosesan yang lebih tinggi dan sering digunakan dalam aplikasi yang lebih kompleks seperti Internet of Things (IoT), sistem robotika, dan perangkat medis. Contoh dari keluarga mikrokontroler 32-bit adalah STM32 dari STMicroelectronics dan ESP32 dari Espressif. Mikrokontroler 32-bit ini sering dilengkapi dengan fitur konektivitas seperti Wi-Fi dan Bluetooth.



Gambar 3. Mikrokontroler 32-bit
(Sumber : www.mouser.co.id)

1.2.4 Mikrokontroler ARM Cortex

Mikrokontroler berbasis ARM adalah yang paling populer dalam keluarga mikrokontroler 32-bit. ARM Cortex-M merupakan seri yang paling umum digunakan dalam aplikasi tertanam berperforma tinggi. ARM memiliki arsitektur yang hemat daya dan sangat efisien dalam menangani berbagai tugas. Contoh populer adalah STM32 yang menggunakan inti ARM Cortex-M.

Tabel 1. Keluarga mikrokontroler

Tipe Mikrokontroler	Contoh	Fitur Utama
8-bit	ATmega328, PIC16	Sederhana, konsumsi daya rendah
16-bit	MSP430	Performansi lebih tinggi, hemat daya
32-bit	STM32, ESP32	Komputasi tinggi, cocok untuk aplikasi IoT
ARM Cortex	STM32, NXP Kinetis	Performansi tinggi, hemat daya, fitur konektivitas
8-bit	ATmega328, PIC16	Sederhana, konsumsi daya rendah

1.3 Aplikasi mikrokontroler di dunia nyata

Mikrokontroler digunakan di berbagai bidang, baik dalam skala kecil seperti proyek DIY maupun dalam aplikasi industri yang besar. Berikut beberapa contoh aplikasi di dunia nyata:

1.3.1 Perangkat Rumah Tangga

Mikrokontroler banyak digunakan dalam peralatan rumah tangga untuk otomatisasi dan kontrol. Contoh-contoh perangkat rumah tangga yang menggunakan mikrokontroler termasuk mesin cuci, oven

microwave, kulkas, dan sistem pengatur suhu (termostat). Di sini, mikrokontroler bertanggung jawab untuk mengendalikan berbagai fungsi seperti waktu operasi, pengaturan suhu, dan manajemen daya.

1.3.2 Sistem Otomotif

Industri otomotif memanfaatkan mikrokontroler untuk berbagai aplikasi, mulai dari sistem kontrol mesin hingga fitur keselamatan seperti airbag dan sistem rem anti-lock (ABS). Mikrokontroler dalam mobil mengontrol fungsi vital seperti pengapian mesin, injeksi bahan bakar, dan bahkan sistem hiburan di dalam mobil.

1.3.3 Industri dan Otomasi

Dalam dunia industri, mikrokontroler digunakan untuk mengotomatisasi proses produksi, mengontrol mesin, dan mengelola sistem kendali proses. Mikrokontroler juga digunakan dalam aplikasi pengendalian motor untuk robotik dan conveyor belt, serta dalam sensor dan alat monitoring seperti pengendali suhu dan tekanan.

1.3.4 Perangkat Kesehatan

Dalam bidang medis, mikrokontroler digunakan untuk berbagai perangkat penting seperti monitor detak jantung, pompa insulin, alat pacu jantung, dan mesin MRI. Mikrokontroler di sini digunakan untuk memproses data sensor dari tubuh pasien dan melakukan pengukuran atau kontrol yang presisi.

1.3.5 Internet of Things (IoT)

Salah satu area aplikasi terbesar saat ini adalah Internet of Things (IoT), di mana mikrokontroler berfungsi sebagai pusat kendali perangkat yang terhubung ke internet. Mikrokontroler seperti ESP32 atau STM32 memungkinkan perangkat untuk berkomunikasi melalui Wi-Fi atau Bluetooth, sehingga dapat dikendalikan dari jarak jauh atau melalui aplikasi cloud. Contoh aplikasinya termasuk lampu pintar, sensor lingkungan, hingga perangkat wearable.

1.3.6 Proyek DIY dan Hobi

Mikrokontroler seperti Arduino dan Raspberry Pi telah membawa revolusi dalam dunia hobi dan proyek DIY. Mereka menyediakan platform yang mudah diprogram dan dilengkapi dengan banyak periferal, sehingga siapa pun bisa membuat perangkat seperti robot, drone, dan perangkat otomatisasi rumah sendiri.

BAB 2

ARSITEKTUR MIKROKONTROLER

Bab ini menjelaskan tentang :

- Komponen utama: CPU, memori, port I/O
- Memahami pin dan periferal mikrokontroler
- Manajemen daya dan sistem clock

2.1 Komponen utama: CPU, memori, port I/O

Mikrokontroler merupakan sebuah sistem komputer sederhana yang dirancang untuk menjalankan tugas tertentu. Terdapat tiga komponen utama dalam arsitektur mikrokontroler, yaitu:

2.1.1 CPU (Central Processing Unit)

CPU (Central Processing Unit) adalah inti dari sebuah mikrokontroler, yang bertugas menjalankan perintah dan mengontrol semua aktivitas yang terjadi di dalam mikrokontroler. CPU dalam mikrokontroler terdiri dari beberapa komponen penting dan bekerja melalui proses pengolahan data yang kompleks, meskipun ukurannya lebih kecil dibandingkan CPU di komputer. Berikut penjelasan lebih detail tentang komponen dan cara kerja CPU di mikrokontroler:

Komponen-Komponen CPU dalam Mikrokontroler

1) ALU (Arithmetic Logic Unit)

- **Fungsi:** ALU adalah bagian dari CPU yang bertanggung jawab untuk melakukan operasi aritmetika (seperti penjumlahan, pengurangan) dan logika (seperti AND, OR, XOR).
- **Operasi Aritmetika:** ALU memungkinkan CPU untuk melakukan operasi dasar seperti penjumlahan dan pengurangan bilangan biner.
- **Operasi Logika:** Operasi logika digunakan untuk keputusan logika dalam pengambilan keputusan program, misalnya memeriksa apakah suatu kondisi benar atau salah (1 atau 0).

2) Register

- **Fungsi:** Register adalah unit penyimpanan kecil yang sangat cepat yang berada di dalam CPU. Register menyimpan data sementara yang sedang diproses oleh CPU.
- **Jenis-Jenis Register:**
 - **Register Data:** Menyimpan data yang sedang diproses oleh ALU.

- **Register Instruksi (Instruction Register):** Menyimpan instruksi yang saat ini sedang dieksekusi oleh CPU.
- **Program Counter (PC):** Menyimpan alamat dari instruksi berikutnya yang akan dieksekusi.
- **Stack Pointer (SP):** Digunakan untuk mengelola tumpukan (stack) memori yang menyimpan data sementara, seperti alamat return selama pemanggilan fungsi.

3) **Control Unit (CU)**

Control Unit bertugas mengontrol semua operasi yang terjadi di dalam CPU dan berkomunikasi dengan bagian-bagian lain dari mikrokontroler.

- **Instruksi Fetch-Decode-Execute:** CU mengarahkan CPU untuk mengambil (fetch) instruksi dari memori, mendekode (decode) instruksi tersebut untuk mengetahui apa yang harus dilakukan, dan kemudian mengeksekusi (execute) instruksi tersebut.
- **Koordinasi Komponen:** CU mengendalikan aliran data antara ALU, register, dan memori serta berkoordinasi dengan perangkat I/O.

4) **Bus**

- **Fungsi:** Bus adalah jalur komunikasi yang menghubungkan komponen-komponen internal CPU, seperti ALU, register, dan CU, dengan komponen eksternal seperti memori dan perangkat I/O.
- **Jenis-Jenis Bus:**
 - **Bus Data:** Membawa data antara CPU dan perangkat lainnya.
 - **Bus Alamat (Address Bus):** Membawa alamat dari data yang akan diproses.
 - **Bus Kontrol:** Membawa sinyal kontrol yang mengatur kapan data harus dibaca atau ditulis ke dalam memori atau perangkat I/O.

2. **Clock:**

- **Fungsi:** Clock adalah sumber sinyal yang mengatur kecepatan eksekusi instruksi dalam CPU. Setiap instruksi dalam CPU dieksekusi berdasarkan siklus clock. Kecepatan clock sangat mempengaruhi kinerja CPU.
- **Siklus Clock:** Siklus clock menentukan berapa banyak waktu yang dibutuhkan untuk mengeksekusi satu instruksi, dan mikrokontroler dapat bekerja dengan clock internal atau menggunakan clock eksternal yang lebih presisi.

Cara Kerja CPU dalam Mikrokontroler

CPU dalam mikrokontroler bekerja dengan mengikuti siklus instruksi yang disebut dengan **Fetch-Decode-Execute**:

1) **Fetch (Mengambil Instruksi):**

Program Counter (PC) menyimpan alamat instruksi berikutnya yang akan dieksekusi. CPU akan mengambil instruksi tersebut dari memori (biasanya dari ROM atau flash memory) dan menyimpannya di Instruction Register.

2) **Decode (Mendekode Instruksi):**

- o Control Unit mendekode instruksi yang telah diambil untuk menentukan jenis operasi yang perlu dilakukan, apakah itu operasi aritmetika, operasi logika, atau pengendalian perangkat I/O.
- o Control Unit juga mengidentifikasi operand yang terlibat, seperti data yang harus diproses oleh ALU atau alamat memori yang perlu diakses.

3) **Execute (Menjalankan Instruksi):**

- o Setelah instruksi didekode, CPU akan menjalankan instruksi tersebut. Jika instruksi adalah operasi aritmetika atau logika, ALU akan mengeksekusi operasi yang diminta.
- o Jika instruksi berhubungan dengan memori, CPU akan membaca atau menulis data dari atau ke memori.
- o Jika instruksi berhubungan dengan perangkat I/O, CPU akan mengirim atau menerima data dari perangkat I/O melalui port yang sesuai.

4) **Update Program Counter (PC):**

Setelah instruksi selesai dieksekusi, PC akan diperbarui untuk menunjuk ke alamat instruksi berikutnya, dan siklus ini akan diulang.

2.1.2 Memori

Memori adalah salah satu komponen penting dalam arsitektur mikrokontroler, karena digunakan untuk menyimpan data dan program yang dijalankan oleh CPU. Pada mikrokontroler, terdapat beberapa jenis memori dengan fungsi yang berbeda, yaitu memori volatil (data hilang ketika daya dimatikan) dan memori non-volatil (data tetap ada meskipun daya dimatikan). Berikut penjelasan lebih detail tentang jenis-jenis memori dan cara kerjanya di dalam mikrokontroler.

a) Jenis-Jenis Memori pada Mikrokontroler

1) RAM (Random Access Memory) – Memori Volatil

- Fungsi: RAM digunakan untuk menyimpan data sementara yang dibutuhkan saat program sedang dijalankan. RAM adalah memori yang bersifat volatile, artinya semua data di dalamnya akan hilang ketika daya pada mikrokontroler dimatikan.
- Kegunaan Utama: RAM biasanya digunakan untuk menyimpan variabel, hasil operasi sementara, atau data yang diperlukan selama program sedang berjalan.
- Stack dan Heap: RAM juga digunakan untuk menyimpan stack dan heap.
 - ✓ Stack: Digunakan untuk menyimpan alamat return saat pemanggilan fungsi, serta variabel lokal yang hanya diperlukan sementara.
 - ✓ Heap: Area memori yang digunakan untuk alokasi dinamis, yaitu data yang dialokasikan dan dibebaskan saat runtime.
- Akses Data Cepat: RAM memiliki akses yang sangat cepat dibandingkan dengan memori lain seperti EEPROM atau flash.

Contoh tipe RAM pada mikrokontroler:
- SRAM (Static RAM): Tipe RAM yang digunakan dalam kebanyakan mikrokontroler. SRAM lebih cepat tetapi lebih mahal dan menggunakan lebih banyak daya daripada DRAM (Dynamic RAM).
- DRAM (Dynamic RAM): Biasanya digunakan pada komputer, tetapi jarang digunakan dalam mikrokontroler karena membutuhkan refresh terus-menerus.

2) ROM (Read-Only Memory) – Memori Non-Volatil

ROM adalah memori yang menyimpan program atau kode yang akan dijalankan oleh mikrokontroler. ROM bersifat non-volatile, artinya data yang disimpan di sini akan tetap ada meskipun daya dimatikan.

Jenis-Jenis ROM

- Mask ROM: Jenis ROM yang diisi dengan data atau program oleh pabrik dan tidak dapat diubah. Biasanya digunakan untuk aplikasi yang tidak memerlukan perubahan program setelah diproduksi.
- PROM (Programmable ROM): Jenis ROM yang dapat diprogram sekali oleh pengguna, tetapi tidak dapat diubah setelah diprogram.
- EPROM (Erasable Programmable ROM): Jenis ROM yang dapat dihapus dan diprogram ulang dengan menggunakan sinar UV.
- EEPROM (Electrically Erasable Programmable ROM): EEPROM dapat dihapus dan diprogram ulang secara elektrik tanpa harus melepaskannya dari mikrokontroler. Ini sangat berguna untuk menyimpan data yang perlu diubah selama operasi, seperti pengaturan konfigurasi atau log data.

3) Flash Memory – Memori Non-Volatil

Flash memory adalah jenis memori non-volatile yang paling umum digunakan pada mikrokontroler modern. Flash memory dapat dihapus dan diprogram ulang secara elektrik seperti EEPROM, tetapi lebih cepat dan lebih efisien dalam hal biaya.

Memori ini digunakan untuk menyimpan firmware, yaitu kode atau program yang akan dijalankan oleh CPU. Banyak mikrokontroler yang menggunakan flash memory untuk menyimpan kode karena dapat diprogram ulang secara mudah tanpa perangkat khusus.

Flash memory lebih efisien daripada EEPROM dalam hal kecepatan penulisan dan penghapusan, serta biasanya memiliki umur yang lebih panjang dalam jumlah siklus baca/tulis yang dapat ditangani.

4) EEPROM (Electrically Erasable Programmable Read-Only Memory) – Memori Non-Volatil

EEPROM adalah jenis memori yang digunakan untuk menyimpan data yang mungkin perlu diubah selama operasi normal perangkat, seperti pengaturan konfigurasi, data

kalibrasi, atau informasi lainnya yang ingin tetap ada setelah mikrokontroler dimatikan.

EEPROM dapat ditulis dan dihapus secara elektrik, meskipun memiliki batas siklus penulisan yang terbatas (biasanya 100.000 hingga 1 juta siklus).

Perbedaan dari Flash Memory: EEPROM lebih lambat dan biasanya memiliki kapasitas lebih kecil dibandingkan dengan flash memory, tetapi EEPROM memungkinkan penulisan byte-per-byte, sedangkan flash memory biasanya ditulis dalam blok.

b) Struktur Penggunaan Memori dalam Mikrokontroler

Pada sebuah mikrokontroler, memori biasanya diorganisasikan menjadi beberapa bagian sesuai dengan fungsi dan kebutuhan operasi. Struktur ini membantu mengelola program dan data dengan cara yang lebih efisien. Berikut adalah beberapa bagian dari pengelolaan memori dalam mikrokontroler:

- **Program Memory** : Program memory, biasanya berupa ROM atau flash memory, digunakan untuk menyimpan kode program atau instruksi yang akan dijalankan oleh CPU. Kode yang ditulis dalam bahasa pemrograman seperti C, Assembly, atau Python (dengan interpreter) diubah menjadi instruksi biner yang disimpan di program memory. Program memory bersifat non-volatile, sehingga program tetap ada bahkan setelah daya dimatikan.
- **Data Memory** : Data memory adalah area RAM yang digunakan untuk menyimpan data-data yang bersifat sementara, seperti variabel dalam program, hasil perhitungan sementara, serta informasi tentang status operasi sistem. Data memory bersifat volatile, sehingga semua data di dalamnya akan hilang ketika daya dimatikan.
- **Stack Memory** : Stack adalah bagian khusus dari RAM yang digunakan untuk menyimpan alamat pengembalian (return address) saat terjadi pemanggilan fungsi atau interrupt. Stack bekerja seperti struktur data LIFO (Last In First Out), di mana data terakhir yang dimasukkan adalah yang pertama diambil. Stack juga

digunakan untuk menyimpan variabel lokal dari fungsi yang dipanggil.

- **Heap Memory** : Heap adalah bagian dari RAM yang digunakan untuk alokasi memori dinamis, yaitu alokasi yang terjadi saat program sedang berjalan. Alokasi ini digunakan untuk menyimpan data besar yang mungkin tidak cukup disimpan dalam stack atau data yang ukuran dan durasi keberadaannya tidak diketahui pada saat kompilasi.
- **EEPROM (Opsional)** : Mikrokontroler tertentu memiliki EEPROM terintegrasi yang dapat digunakan untuk menyimpan data jangka panjang yang dapat dimodifikasi selama operasi, seperti pengaturan konfigurasi pengguna atau nilai kalibrasi sensor.

c) Peran Memori dalam Mikrokontroler

Memori dalam mikrokontroler memainkan peran krusial dalam berbagai aspek, termasuk:

- 1) Menyimpan Program: Memori flash atau ROM digunakan untuk menyimpan firmware atau perangkat lunak yang akan dijalankan oleh mikrokontroler.
- 2) Menyimpan Data: RAM menyimpan data sementara yang digunakan selama program berjalan. EEPROM atau flash digunakan untuk menyimpan data permanen yang perlu disimpan ketika perangkat dimatikan.
- 3) Mengelola Fungsi Sistem: Memori stack dan heap digunakan untuk mengelola pemanggilan fungsi, interrupt, dan alokasi memori dinamis saat runtime. Port I/O (Input/Output)
Port I/O memungkinkan mikrokontroler untuk berinteraksi dengan perangkat eksternal. Perangkat ini bisa berupa sensor, saklar, LED, atau motor. Port I/O dapat dikonfigurasi sebagai input untuk menerima sinyal dari dunia luar atau sebagai output untuk mengontrol perangkat.

2.2 Memahami pin dan periferal mikrokontroler

Mikrokontroler memiliki sejumlah pin dan periferal yang mendukung fungsinya. Berikut adalah penjelasan dasar tentang pin dan periferal umum:

2.2.1 Pin Input/Output (GPIO)

General Purpose Input/Output (GPIO) adalah pin serbaguna yang dapat diatur sebagai input atau output. Sebagai input, GPIO menerima sinyal dari perangkat eksternal, seperti tombol atau sensor. Sebagai output, GPIO dapat mengendalikan perangkat, seperti LED atau relay.

2.2.2 Pin VCC dan GND

Pin ini digunakan untuk menyuplai tegangan ke mikrokontroler. VCC adalah pin untuk tegangan positif (biasanya 3.3V atau 5V), dan GND adalah pin ground.

2.2.3 Pin PWM (Pulse Width Modulation)

Pin PWM memungkinkan mikrokontroler untuk menghasilkan sinyal analog dari keluaran digital dengan cara mengubah rasio on/off dari sebuah sinyal digital. Pin ini sering digunakan untuk mengontrol kecerahan LED atau kecepatan motor.

2.2.4 Pin ADC (Analog to Digital Converter)

ADC mengubah sinyal analog menjadi data digital yang dapat dibaca oleh mikrokontroler. Pin ini digunakan untuk membaca sensor analog, seperti potensiometer atau sensor suhu.

2.2.5 Periferal UART, SPI, dan I2C

Periferal komunikasi seperti UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), dan I2C (Inter-Integrated Circuit) digunakan untuk berkomunikasi dengan perangkat eksternal, seperti modul sensor, layar LCD, atau modul komunikasi lain.

2.3 Manajemen daya dan sistem clock

Mikrokontroler memiliki mekanisme manajemen daya dan sistem clock untuk mengoptimalkan kinerja dan efisiensi energi. Berikut adalah komponen dan teknik yang berperan:

2.3.1 Manajemen Daya

- **Mode Sleep:** Mikrokontroler memiliki beberapa mode sleep yang mengurangi konsumsi daya saat tidak digunakan. Mode sleep memungkinkan bagian-bagian tertentu dari mikrokontroler untuk dimatikan sementara waktu, sehingga menghemat energi.
- **Low-Power Peripherals:** Beberapa mikrokontroler dilengkapi dengan periferal yang dapat tetap aktif selama mode sleep,

memungkinkan perangkat untuk tetap berfungsi dengan konsumsi daya yang sangat rendah.

2.3.2 Sistem Clock

- **Clock Internal dan Eksternal:** Mikrokontroler menggunakan sinyal clock untuk menentukan kecepatan eksekusi instruksi. Clock dapat berasal dari sumber internal (internal oscillator) atau eksternal (crystal oscillator). Crystal oscillator biasanya lebih presisi dibandingkan dengan oscillator internal.
- **Prescaler:** Prescaler adalah mekanisme yang mengubah kecepatan clock dengan cara membagi frekuensi clock yang digunakan oleh mikrokontroler. Dengan menurunkan kecepatan clock, konsumsi daya bisa dikurangi, namun kinerja juga akan melambat.

Sistem manajemen daya dan clock sangat penting terutama pada aplikasi berbasis mikrokontroler yang menggunakan sumber daya terbatas, seperti perangkat yang ditenagai oleh baterai.

BAB 3

PEMROGRAMAN MIKROKONTROLER

Bab ini menjelaskan tentang :

- Arduino
- Bahasa pemrograman
- Lingkungan pengembangan
- Menulis dan mengunggah program dasar

3.1 Arduino

Arduino adalah pengendali mikro single-board yang bersifat open-source, diturunkan dari Wiring platform, dirancang untuk memudahkan penggunaan elektronik dalam berbagai bidang. Hardwarenya memiliki prosesor Atmel AVR dan softwrenya memiliki bahasa pemrograman sendiri.

Arduino juga merupakan platform hardware terbuka yang ditujukan kepada siapa saja yang ingin membuat purwarupa peralatan elektronik interaktif berdasarkan hardware dan software yang fleksibel dan mudah digunakan. Mikrokontroler diprogram menggunakan bahasa pemrograman arduino yang memiliki kemiripan syntax dengan bahasa pemrograman C. Karena sifatnya yang terbuka maka siapa saja dapat mengunduh skema hardware arduino dan membangunnya.

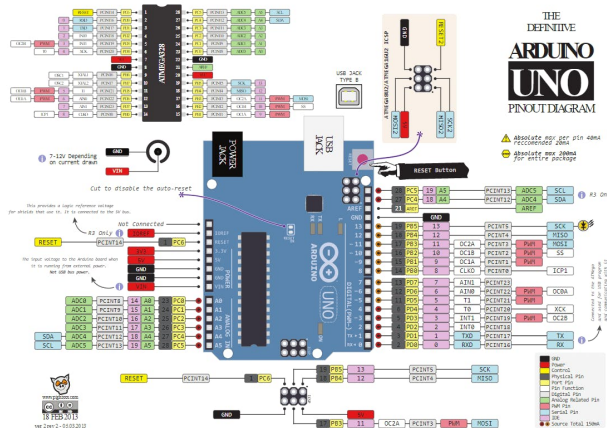
Arduino menggunakan keluarga mikrokontroler ATMEga yang dirilis oleh Atmel sebagai basis, namun ada individu/perusahaan yang membuat clone arduino dengan menggunakan mikrokontroler lain dan tetap kompatibel dengan arduino pada level hardware. Untuk fleksibilitas, program dimasukkan melalui bootloader meskipun ada opsi untuk mem-bypass bootloader dan menggunakan downloader untuk memprogram mikrokontroler secara langsung melalui port ISP.

Ada beberapa jenis arduino diantaranya adalah :

3.1.1 Arduino UNO

Jenis Arduino yang paling sering digunakan. Terutama untuk pemula atau media pembelajaran sangat disarankan menggunakan Arduino Uno. Selain banyaknya referensi yang membahas jenis arduino yang satu ini, juga karena chip mikrokontroller yang digunakan memakai jenis DIL / DIP (Dual In-Line Package). Sangat memudahkan pengguna mengganti chip mikrokontroller, jika terjadi kerusakan, dan juga compatible dengan banyak Shield tambahan seperti, Ethernet, SD-CARD, GSM,dll. Versi yang terakhir adalah

Arduino uno R3 (Revisi 3), menggunakan chip mikrokontroler Atmel AVR ATMEGA328, memiliki 14 pin I/O digital (6 diantaranya pin PWM), 6 pin input analog, . Komunikasi USB A to USB B (USB Printer) memudahkan komunikasi hardware dengan perangkat komputer / laptop.



Gambar 4. Pinout diagram arduino uno
(Sumber : <https://www.circuito.io>)

Tabel 2. Spesifikasi Arduino Uno

No	Items	Specifications
1	Microprosesor	Atmega328P
2	Freq Clock	Up to 16 MHz
3	SRAM	2 Kb
4	Flash memori	32 Kb
5	EEPROM	1 Kb
6	GPIO	20 Pin
	Chn Digital	14 Pin
	Chn Analog	4 Pin
	Chn PWM	6 Pin

(Sumber : https://id.wikipedia.org/wiki/Arduino_Uno)

3.1.2 Arduino Mega

Arduino mega 2560 adalah papan mikrokontroler ATmega2560 berdasarkan (datasheet) memiliki 54 digital pin input / output (dimana

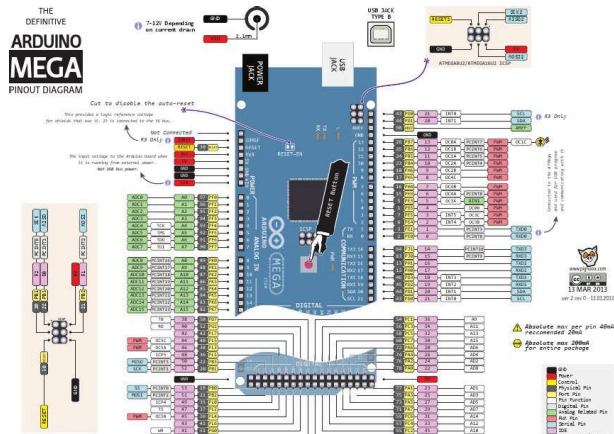
15 dapat digunakan sebagai output PWM), 16 analog input, 4 UART (hardware port serial), osilator kristal 16 MHz, koneksi USB, jack listrik, header ICSP, dan tombol reset. Ini berisi semua yang diperlukan untuk mendukung mikrokontroler, hanya menghubungkannya ke komputer dengan kabel USB atau power dengan adaptor AC-DC atau baterai. Arduino Mega kompatibel dengan sebagian besar shield, dirancang untuk Arduino Duemilanove atau Diecimila. Arduino Mega2560 berbeda dari semua board sebelumnya, tidak menggunakan chip driver FTDI USB-to-serial. Sebaliknya, fitur ATmega16U2 (ATmega8U2 dalam revisi 1 dan revisi 2 papan) diprogram sebagai konverter USB-to-serial. Revisi 2 dewan Mega2560 memiliki resistor menarik garis 8U2 HWB ke tanah, sehingga lebih mudah untuk dimasukkan ke dalam mode DFU.

Revisi 3 memiliki fitur-fitur baru yaitu menambahkan pin SDA dan pin SCL yang dekat dengan pin AREF dan dua pin baru lainnya ditempatkan dekat dengan pin RESET, yang IOREF yang memungkinkan *shield* untuk beradaptasi dengan tegangan yang tersedia dari *board*. Di masa depan, *shield* akan kompatibel baik dengan sistem yang menggunakan AVR yang beroperasi dengan 5V dan dengan Arduino Due yang beroperasi dengan 3.3V.

Tabel 3. Spesifikasi Arduino Mega

No	Items	Specifications
1	Microprosesor	Atmega2560
2	Freq Clock	Up to 16 MHz
3	SRAM	8 Kb
4	Flash memori	256 Kb
5	EEPROM	4 Kb
6	GPIO	56 Pin
	Chn Digital	38 Pin
	Chn Analog	16 Pin
	Chn PWM	15 Pin

(Sumber : <https://store.arduino.cc/>)



Gambar 5. Pinout diagram arduino mega
(Sumber : <https://saptaji.com/>)

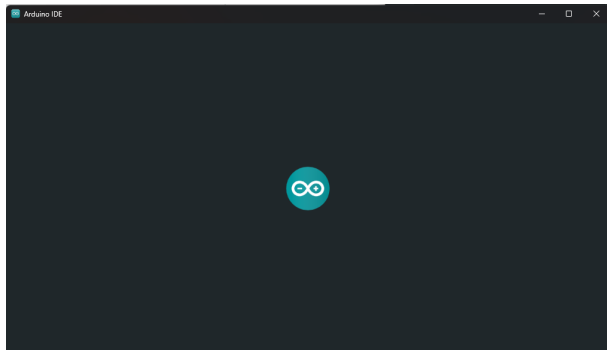
3.2 Bahasa pemrograman

Bahasa pemrograman Arduino adalah bahasa berbasis C/C++ yang disederhanakan agar lebih mudah digunakan oleh pemula dalam dunia mikrokontroler. Bahasa ini dirancang khusus untuk mempermudah pengguna dalam mengontrol perangkat keras seperti sensor, motor, dan LED. Arduino IDE menyediakan serangkaian fungsi yang memudahkan penulisan kode untuk berbagai keperluan, seperti mengatur pin sebagai input atau output, membaca nilai dari sensor, dan mengontrol perangkat eksternal. Arduino juga memiliki fungsi-fungsi khusus untuk komunikasi, baik melalui USB maupun protokol seperti I2C dan SPI, yang memungkinkan mikrokontroler untuk berinteraksi dengan perangkat lain. Dengan menggunakan bahasa ini, pengguna dapat membuat program yang secara otomatis merespons perubahan lingkungan, mengontrol aktuator, dan menjalankan logika tertentu dalam aplikasi dunia nyata. Karena berbasis C/C++, bahasa ini memiliki struktur yang mirip dengan bahasa pemrograman lain yang banyak digunakan, sehingga memudahkan pembelajaran lebih lanjut dalam bidang pemrograman dan elektronika.

3.3 Lingkungan pengembangan

Lingkungan pengembangan arduino, yang dikenal sebagai Arduino IDE (Integrated Development Environment), dirancang untuk menyederhanakan penulisan kode dan pengunggahan program, sehingga mudah diakses oleh

pemula maupun pengguna yang lebih berpengalaman. IDE ini menyediakan editor tempat pengguna dapat menulis, mengedit, dan menyimpan kode dalam bahasa pemrograman Arduino..



Gambar 6. Arduino IDE 2
(Sumber : <https://docs.arduino.cc/>)

Arduino IDE 2 memiliki sidebar baru, membuat alat yang paling umum digunakan lebih mudah diakses.



Gambar 7. Menu Arduino IDE 2
(Sumber : <https://docs.arduino.cc/>)

- **Verify / Upload** - kompilasi dan unggah kode Anda ke Papan Arduino Anda.
- **Pilih Board & Port** - mendeteksi papan Arduino secara otomatis muncul di sini, bersama dengan nomor port.
- **Sketchbook** - di sini Anda akan menemukan semua sketsa Anda secara lokal disimpan di komputer Anda. Selain itu, Anda dapat

sinkron dengan Arduino Cloud, dan juga mendapatkan sketsa Anda dari lingkungan online.

- **Board Manager** - telusuri melalui paket Arduino & pihak ketiga yang dapat dipasang. Misalnya, menggunakan papan Arduino UNO membutuhkan Arduino UNO paket yang dipasang.
- **Library Manager** - menelusuri ribuan perpustakaan Arduino, yang dibuat oleh Arduino & komunitasnya.
- **Debugger** - tes dan debug program secara real time.
- **Search** - mencari kata kunci dalam kode Anda.
- **Open Serial Monitor** - membuka alat Serial Monitor, sebagai tab baru di konsol.

3.4 Menulis dan mengunggah program dasar

3.4.1 Struktur dasar program arduino

Program Arduino selalu terdiri dari dua fungsi utama: `setup()` dan `loop()`. Ini adalah dua komponen utama dalam setiap program Arduino dan keduanya memiliki peran khusus.

- ✓ Fungsi **`setup()`** adalah tempat di mana Anda menginisialisasi semua pengaturan dan konfigurasi awal untuk Arduino. Fungsi ini akan dijalankan sekali saja ketika Arduino pertama kali dinyalakan atau di-reset. Biasanya, di dalam `setup()` kita menulis pengaturan dasar seperti inisialisasi pin sebagai INPUT atau OUTPUT (menggunakan fungsi `pinMode()`), atau memulai komunikasi serial dengan `Serial.begin()`.



CPP

```
void setup() {  
    // Mengatur pin LED bawaan sebagai output  
    pinMode(LED_BUILTIN, OUTPUT);  
    // Memulai komunikasi serial dengan baud rate 9600  
    Serial.begin(9600);  
}
```

- ✓ Fungsi **loop()** adalah inti dari program Arduino yang berisi instruksi yang akan terus berulang setelah **setup()** selesai dijalankan. Ini memungkinkan program untuk melakukan tugas berulang seperti membaca sensor atau mengaktifkan aktuator secara terus-menerus. Fungsi **loop()** ini seperti “detak jantung” Arduino karena terus berjalan selama Arduino masih menyala.



CPP

```
void loop() {  
    // Menyalakan LED  
    digitalWrite(LED_BUILTIN, HIGH);  
    // Menunggu selama 1 detik  
    delay(1000);  
    // Mematikan LED  
    digitalWrite(LED_BUILTIN, LOW);  
    // Menunggu selama 1 detik  
    delay(1000);  
}
```

3.4.2 Sintaks dasar

Sintaks dasar pemrograman Arduino meliputi aturan penulisan kode yang diadaptasi dari bahasa C/C++, yang memungkinkan pemrogram menulis instruksi yang dipahami oleh mikrokontroler. Adapun aturan itu adalah :

- Pernyataan dan Baris Kode: Setiap instruksi atau pernyataan diakhiri dengan tanda titik koma (;). Ini penting karena tidak adanya tanda ; dapat menyebabkan kesalahan.
- Blok Kode: Blok kode ditulis di antara kurung kurawal {}. Kode di dalam **setup()** dan **loop()** selalu berada di antara kurung ini.
- Komentar: Menggunakan komentar sangat membantu dalam menjelaskan bagian kode. Komentar satu baris ditulis dengan **//**, sedangkan komentar beberapa baris ditulis di antara **/* ... */**.



CPP

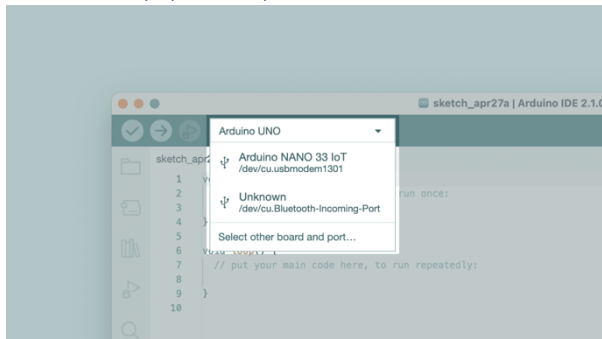
```
// Ini adalah komentar satu baris  
/* Ini adalah komentar  
beberapa baris */
```

d. Pengaturan pin digital dan analog

- 1) Pin digital pada Arduino dapat diatur sebagai INPUT atau OUTPUT menggunakan fungsi `pinMode(pin, mode)`;
- 2) `digitalWrite(pin, value)`; digunakan untuk mengirim sinyal HIGH (nyala) atau LOW (mati) ke pin digital yang diatur sebagai OUTPUT.
- 3) `digitalRead(pin)`; digunakan untuk membaca nilai dari pin digital yang diatur sebagai INPUT.
- 4) `analogRead(pin)`; membaca nilai analog yang berkisar antara 0 hingga 1023 pada sebagian besar Arduino.
- 5) `analogWrite(pin, value)`; (hanya pada pin yang mendukung PWM) mengatur nilai keluaran PWM antara 0 hingga 255, yang berguna untuk mengontrol intensitas LED atau kecepatan motor.

3.5 Mengunggah program dasar

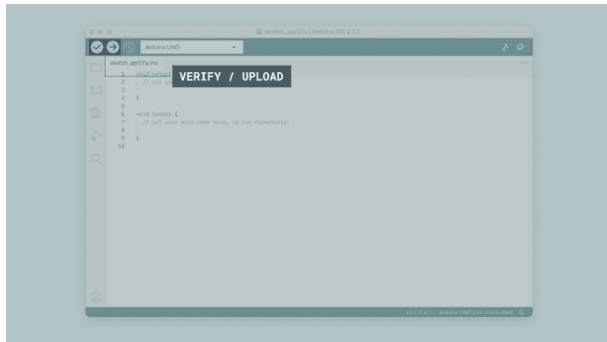
Sebelum mengunggah program dasar, kita harus memilih papan apa yang kita unggah. Selain tombol verifikasi dan unggah Anda harus melihat menu drop down, ini akan, dalam banyak kasus, menampilkan papan Arduino yang terhubung ke komputer Anda. Jika papan Anda tidak terdeteksi secara otomatis, Anda dapat menekan "Pilih papan dan port lain ..." dalam drop-down dan ikuti petunjuk, atau pergi ke Tools > Board and Tools > Port di menu toolbar untuk memilih papan dan port secara manual.



Gambar 8. Memilih *board* yang sesuai

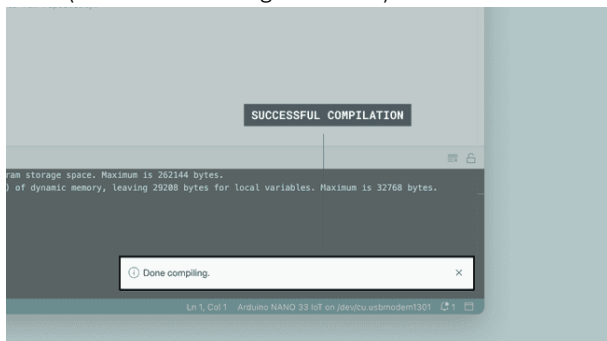
(Sumber : <https://docs.arduino.cc/>)

Pada menu sebelah kiri, ada tanda centang dan panah yang mengarah ke kanan. Tanda centang digunakan untuk memverifikasi, dan panah digunakan untuk mengunggah program.



Gambar 9. Tombol verifikasi dan unggah
(Sumber : <https://docs.arduino.cc/>)

Klik tombol verifikasi (verify). Setelah beberapa detik, kita dapat melihat hasil aksi di terminal (kotak hitam di bagian bawah).



Gambar 10. Kompilasi berhasil
(Sumber : <https://docs.arduino.cc/>)

Dengan board yang sudah dipilih, klik tombol Upload untuk mulai mengunggah sketsa ke board.

Setelah proses selesai, pemberitahuan akan muncul di bagian kanan bawah jendela IDE. Terkadang ada beberapa komplikasi saat mengunggah, dan jika terjadi kesalahan, pesan kesalahan tersebut juga akan ditampilkan di sini.

BAB 4

INTERFACING DENGAN SENSOR

Bab ini menjelaskan tentang :

- Sensor dan prinsip kerja
- Jenis sensor
- Membaca sinyal analog dan digital
- Contoh praktis: Menghubungkan dan membaca data dari sensor DHT22

4.1 Sensor dan prinsip kerja

Sensor adalah perangkat yang berfungsi untuk mendeteksi atau mengukur suatu perubahan fisik dalam lingkungan atau objek, kemudian mengubahnya menjadi sinyal listrik atau data yang dapat dibaca oleh manusia atau sistem elektronik. Dengan kata lain, sensor bekerja sebagai "indera" bagi perangkat elektronik untuk merasakan dan memahami kondisi di sekitarnya. Sensor ini banyak digunakan dalam berbagai bidang, seperti otomasi industri, kesehatan, pertanian, dan perangkat rumah pintar.

Setiap jenis sensor memiliki prinsip kerja yang berbeda, tergantung pada perubahan fisik yang ingin dideteksi. Berikut adalah langkah umum cara kerja sensor:

1. **Pendeteksian Perubahan Fisik:** Sensor pertama-tama mendeteksi perubahan fisik tertentu, seperti suhu, tekanan, kelembaban, cahaya, atau gerakan. Ini dilakukan melalui elemen sensitif yang merespons terhadap perubahan tersebut. Misalnya, sensor suhu memiliki elemen yang sensitif terhadap panas, sementara sensor cahaya memiliki elemen yang peka terhadap cahaya.
2. **Konversi ke Sinyal Listrik:** Setelah mendeteksi perubahan fisik, sensor mengonversinya menjadi sinyal listrik. Proses ini biasanya melibatkan perubahan pada elemen tertentu dalam sensor, seperti resistansi, kapasitansi, atau induktansi, tergantung pada jenis sensor yang digunakan. Misalnya, pada sensor suhu termistor, resistansi berubah dengan perubahan suhu. Perubahan ini kemudian diubah menjadi sinyal listrik.
3. **Pemrosesan Sinyal:** Sinyal listrik yang dihasilkan kemudian diproses untuk mendapatkan data yang lebih akurat atau sesuai dengan

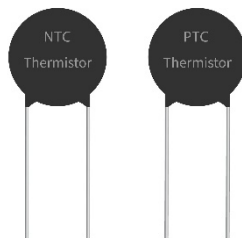
kebutuhan. Sinyal dapat diperkuat, disaring, atau diubah menjadi bentuk digital agar lebih mudah dibaca oleh mikrokontroler atau sistem komputer. Banyak sensor modern memiliki rangkaian pemrosesan sinyal internal yang langsung menghasilkan output digital.

4. Pengiriman Data ke Mikrokontroler atau Komputer: Setelah sinyal diproses, data ini kemudian dikirimkan ke perangkat pemrosesan, seperti mikrokontroler (misalnya, Arduino) atau komputer. Mikrokontroler akan membaca data dari sensor dan menjalankan instruksi yang sesuai. Misalnya, jika sensor mendeteksi suhu tinggi, mikrokontroler dapat memicu sistem pendingin atau memberi peringatan.
5. Interpretasi dan Respons: Data yang diterima oleh mikrokontroler atau komputer kemudian diinterpretasikan dan direspons sesuai kebutuhan aplikasi. Jika sistem mendeteksi perubahan yang sesuai dengan ambang batas tertentu, perangkat dapat melakukan tindakan otomatis, seperti menyalakan alarm, mengaktifkan relay, atau menyimpan data untuk analisis lebih lanjut.

4.2 Jenis sensor

4.2.1 Sensor suhu

Sensor suhu digunakan untuk mendeteksi atau mengukur suhu lingkungan atau suatu objek. Sebagai contoh thermistor adalah resistor yang resistansinya berubah seiring dengan perubahan suhu. Saat suhu naik, resistansi thermistor akan menurun (pada thermistor tipe NTC). Arduino atau mikrokontroler kemudian mengukur tegangan yang dihasilkan dari perubahan resistansi ini, lalu mengonversinya menjadi nilai suhu menggunakan rumus yang sesuai.



Gambar 11. Sensor suhu
(Sumber : <https://www.wattco.com/>)

Jenis-Jenis Sensor Suhu:

- ✓ DHT11/DHT22: Sensor suhu dan kelembaban sederhana yang memberikan keluaran digital. DHT22 lebih akurat dibandingkan DHT11.
- ✓ LM35: Sensor suhu analog dengan sensitivitas linier; menghasilkan tegangan yang sesuai dengan suhu dalam derajat Celsius.
- ✓ DS18B20: Sensor suhu digital yang menggunakan protokol 1-Wire, sehingga dapat digunakan dengan beberapa sensor di satu pin data.

4.2.2 Sensor Gas

Fungsi utama sensor gas adalah mendeteksi keberadaan dan konsentrasi gas tertentu di lingkungan sekitarnya. Sensor ini digunakan untuk berbagai keperluan, mulai dari keamanan, kesehatan, hingga otomasi industri.

Beberapa sensor gas dapat memberikan output berupa konsentrasi gas dalam satuan ppm (parts per million) sebagai contoh adalah MQ7 untuk karbon monoksida (CO) dan MH-Z19B untuk karbon dioksida (CO₂).



Gambar 12. Sensor MQ-7
(Sumber : <https://robocraze.com/>)

4.2.3 Sensor Gerak

Sensor gerak memiliki fungsi utama untuk mendeteksi keberadaan dan pergerakan objek atau manusia di sekitarnya. Sensor ini digunakan dalam berbagai aplikasi seperti keamanan, otomasi, dan penghematan energi. Sebagai contoh sensor *passive infrared receiver* (PIR) yang bekerja dengan cara mendeteksi perubahan sinar inframerah pasif akibat pergerakan tubuh.

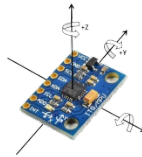


Gambar 13. Sensor PIR

(Sumber : <https://www.edukasielektronika.com/>)

4.2.4 Sensor Posisi dan Percepatan

Sensor posisi dan percepatan memiliki fungsi utama untuk mendeteksi posisi, gerakan, orientasi, dan perubahan kecepatan suatu objek. Sensor ini sangat berguna dalam berbagai aplikasi, mulai dari perangkat elektronik portabel hingga sistem otomasi dan robotika. Sebagai contoh potensiometer digunakan untuk mendeteksi posisi sudut pada tuas atau mekanisme rotasi, sensor MPU6050 (accelerometer) digunakan untuk mendeteksi getaran atau guncangan.

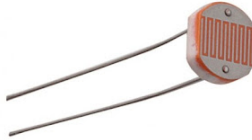


Gambar 14. Sensor accelerometer MPU6050

(Sumber : <https://github.com/>)

4.2.5 Sensor Optik

Sensor optik adalah perangkat yang berfungsi untuk mendeteksi cahaya atau perubahan karakteristik optik, seperti intensitas, warna, atau pola cahaya, dan mengubahnya menjadi sinyal listrik. Sensor ini digunakan dalam berbagai aplikasi industri, medis, otomasi, dan teknologi sehari-hari. Sebagai contoh sensor *light dependent resistor (LDR)* adalah komponen elektronik berupa resistor yang nilai resistansinya berubah sesuai dengan intensitas cahaya yang diterimanya.



Gambar 15. Sensor *light dependent resistor (LDR)*

(Sumber : <https://www.arduinoindonesia.id/>)

4.2.6 Sensor Aliran

Sensor aliran adalah perangkat yang digunakan untuk mendeteksi, mengukur, atau memantau kecepatan atau volume aliran fluida (cairan atau gas) dalam suatu sistem. Sensor ini sering digunakan dalam aplikasi industri, otomotif, kesehatan, dan rumah tangga untuk mengontrol dan mengoptimalkan aliran fluida.



Gambar 16. *Flow sensor*

(Sumber : <https://www.arduinoindonesia.id/>)

4.2.7 Sensor Biometrik

Sensor biometrik adalah perangkat yang digunakan untuk mengukur, mengenali, dan memverifikasi identitas seseorang berdasarkan karakteristik biologis atau perilaku unik mereka. Sensor ini memainkan peran penting dalam keamanan, pengendalian akses, dan aplikasi identitas digital.



Gambar 17. *Fingerprint*

(Sumber : <https://www.makerlab-electronics.com/>)

4.3 Membaca sinyal analog dan digital

Pada mikrokontroler, sinyal digital dan analog adalah jenis sinyal yang berbeda dalam cara kerja dan penggunaannya:

4.3.1 Sinyal Digital

Sinyal digital adalah jenis sinyal yang hanya memiliki dua keadaan: HIGH (1) dan LOW (0). Mikrokontroler menggunakan sinyal digital untuk membaca status perangkat seperti tombol, saklar, atau sensor tertentu. Membaca sinyal digital melibatkan deteksi tegangan pada pin input, yang biasanya dibandingkan dengan ambang batas logika tertentu.

Pemahaman mendalam tentang pembacaan sinyal digital sangat penting dalam desain dan implementasi sistem elektronik berbasis mikrokontroler. Artikel ini akan menjelaskan secara rinci konsep, perangkat yang dibutuhkan, cara kerja, hingga implementasi pembacaan sinyal digital pada mikrokontroler.

Cara Kerja Pembacaan Sinyal Digital

Dalam sistem mikrokontroler seperti Arduino, konfigurasi pin input dilakukan menggunakan perintah tertentu, misalnya *pinMode(pin, INPUT)*. Setelah dikonfigurasi, mikrokontroler akan mendeteksi level tegangan pada pin tersebut; tegangan yang mendekati nilai *Vcc* diinterpretasikan sebagai logika HIGH, sedangkan tegangan yang mendekati GND diinterpretasikan sebagai logika LOW. Agar pembacaan tegangan stabil, diperlukan resistor eksternal berupa resistor *pull-up* yang terhubung ke *Vcc* untuk menetapkan nilai *default* HIGH, atau resistor *pull-down* yang terhubung ke GND untuk menetapkan nilai *default* LOW. Peran resistor ini sangat penting guna mencegah terjadinya sinyal "mengambang" (*floating*) saat perangkat input tidak aktif.

Konsep input digital ini umumnya diterapkan pada pembacaan status tombol untuk mengontrol berbagai fungsi perangkat. Pada skema dasarnya, tombol fisik dihubungkan langsung ke salah satu pin digital mikrokontroler. Menariknya, mikrokontroler modern seperti Arduino telah dilengkapi fitur resistor *pull-up* internal di dalam chip, sehingga pengguna dapat mengaktifkannya secara praktis melalui perintah `INPUT_PULLUP` tanpa perlu menambahkan resistor fisik secara terpisah.

```
#define tombol 2 // Pin digital untuk tombol

void setup() {
  pinMode(tombol, INPUT_PULLUP); // input dengan pull-up internal
  Serial.begin(9600); // Memulai komunikasi serial
}

void loop() {
  int statusTombol = digitalRead(tombol); // Membaca status tombol
  if (statusTombol == LOW) { // Tombol ditekan (karena pull-up default HIGH)
    Serial.println("Tombol ditekan");
  } else {
    Serial.println("Tombol tidak ditekan");
  }
  delay(100); // Penundaan untuk mengurangi pembacaan berlebihan
}
```

4.3.2 Sinyal Analog: Sinyal yang memiliki nilai kontinu dalam rentang tertentu, biasanya 0–Vref (tegangan referensi mikrokontroler).

Sinyal analog adalah jenis sinyal yang merepresentasikan data dalam bentuk kontinu. Ini berarti nilainya bisa berubah secara halus dan bertahap, tidak melompat langsung dari satu nilai ke nilai lainnya seperti sinyal digital.

Karakteristik sinyal analog:

- Kontinu: Tidak ada diskrit atau jeda di antara perubahan nilai.
- Tak Terbatas dalam Detail: Dalam rentang tertentu, terdapat jumlah nilai yang tak terhingga.
- Dipengaruhi Lingkungan: Sinyal analog sering digunakan untuk merepresentasikan fenomena dunia nyata seperti suara, cahaya, suhu, dan tekanan.

Contoh dalam kehidupan nyata :

- ✓ Suara manusia, Getaran udara (amplitudo berubah terus)
- ✓ Suhu ruangan, Perubahan temperatur yang halus
- ✓ Cahaya matahari, Intensitas sinar yang bertambah atau berkurang secara bertahap.

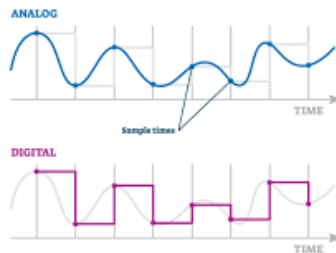
Mengapa Analog? Karena dunia nyata hampir seluruhnya beroperasi dalam bentuk kontinu, sehingga untuk menangkap realitas

secara akurat, kita memerlukan sistem yang mampu membaca sinyal analog.

Tabel 4. Perbandingan sinyal analog vs sinyal digital

Sinyal analog	Sinyal digital
Nilai kontinu (tak terbatas)	Nilai diskrit (terbatas, 0 atau 1)
Representasi halus dan alami	Representasi berbentuk kotak (step-step)
Contoh: Suara manusia	Contoh: Data komputer

Berikut adalah gambar yang menjelaskan perbedaan antara sinyal analog dan digital:



Gambar 18. Bentuk sinyal analog dan digital

(Sumber : <https://crossroadenergy-com/>)

Pada gambar di atas, kurva biru (analog) menunjukkan perubahan nilai yang halus dan kontinu sedangkan kurva merah (digital) menunjukkan perubahan nilai yang diskrit, berupa loncatan antara dua tingkat tegangan.

Cara Kerja Pembacaan Sinyal Analog

Mikrokontroler tidak dapat membaca sinyal analog secara langsung dalam bentuk aslinya. Oleh karena itu, digunakan sebuah modul yang disebut Analog-to-Digital Converter (ADC).

Prosesnya sebagai berikut :

Sampling : Mengambil nilai sinyal analog pada waktu tertentu.

Quantization : Mengubah nilai sampling menjadi nilai digital dalam bit tertentu (misalnya 10-bit ADC menghasilkan nilai dari 0 hingga 1023).

Encoding : Mentransformasikan hasil kuantisasi menjadi format digital biner.

Semakin tinggi resolusi ADC, semakin akurat pembacaan nilai sinyal analog.

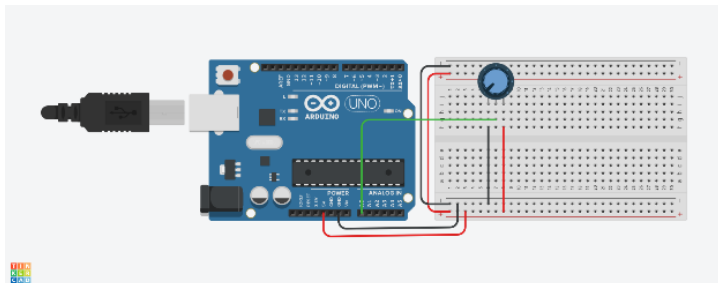
Contoh: Pada Arduino Uno, ADC memiliki resolusi 10-bit, sehingga:

- 0 Volt dibaca sebagai 0
- 5 Volt dibaca sebagai 1023
- 2,5 Volt kira-kira dibaca sebagai 512

Contoh Pembacaan Variable Resistor (Potensiometer) pada Arduino.

Potensiometer adalah resistor variabel yang dapat mengubah nilai hambatan listrik secara manual. Biasanya digunakan untuk memberikan input analog ke mikrokontroler.

Rangkaian sederhana pembacaan sinyal analog (potensiometer) dengan menggunakan arduino UNO.



Gambar 19. Rangkaian pembeacaan sensor analog

(Sumber : <https://www.tinkercad.com>)

Pada gambar di atas, pin tengah potensiometer ke pin analog A0 pada arduino UNO. dua pin sisi lain ke VCC (5V) dan GND.

```
int potPin = A0; // Pin analog A0
int potValue = 0; // Variabel untuk menyimpan nilai potensiometer

void setup() {
  Serial.begin(9600); // Inisialisasi komunikasi serial
}
```

```

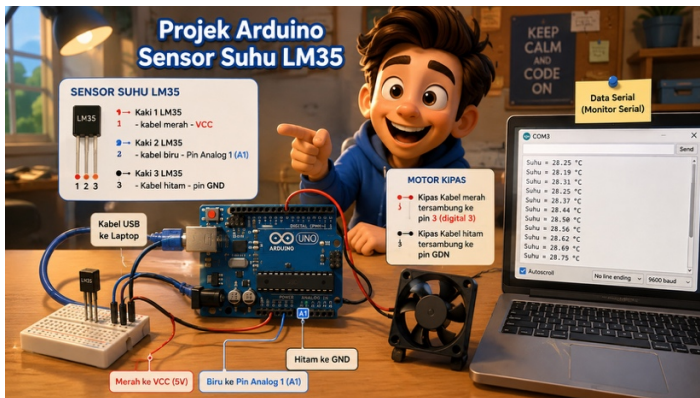
void loop() {
    potValue = analogRead(potPin); // Membaca nilai analog
    Serial.println(potValue);      // Menampilkan nilai ke Serial Monitor
    delay(500);                   // Delay 500 ms
}

```

Faktor-Faktor yang Mempengaruhi Pembacaan Sinyal Analog

- Kebersihan Sinyal: Noise (gangguan) dapat mempengaruhi pembacaan sinyal analog. Gunakan kapasitor atau teknik filtering jika perlu.
- Referensi Tegangan (Vref): ADC membandingkan sinyal input terhadap tegangan referensi. Jika Vref berubah-ubah, hasil pembacaan juga akan berubah.
- Kecepatan Sampling: Untuk sinyal yang cepat berubah, sampling rate harus cukup tinggi agar pembacaan tidak ketinggalan perubahan sinyal.

4.4 Contoh praktis: Menghubungkan dan membaca data dari sensor DHT22



Gambar 20. Contoh praktis sensor LM35

Wiring diagram pada gambar tersebut menghubungkan Arduino Uno, Sensor Suhu LM35, dan sebuah Motor Kipas (Fan DC).

Berikut rincian koneksi kabelnya:

1. Koneksi Sensor Suhu LM35 (ke Arduino)
Dilihat dari sisi depan/datar LM35 (urutan pin 1, 2, 3 dari kiri ke kanan):

- Pin 1 (VCC) → Dihubungkan dengan kabel merah ke pin 5V / VCC pada Arduino (sebagai sumber tegangan sensor).
 - Pin 2 (Output Analog) → Dihubungkan dengan kabel biru ke pin A1 (Analog In 1) pada Arduino (untuk membaca data suhu).
 - Pin 3 (GND) → Dihubungkan dengan kabel hitam ke pin GND pada Arduino (sebagai ground/negatif).
2. Koneksi Motor Kipas (ke Arduino)
- Kabel Positif (Merah) → Dihubungkan ke pin Digital 3 (D3) pada Arduino (digunakan sebagai pin kontrol sakelar atau sinyal PWM kipas).
 - Kabel Negatif (Hitam) → Dihubungkan ke pin GND pada Arduino.
3. Komunikasi & Daya Sistem
- Kabel USB Arduino → Dicolokkan ke port USB laptop/PC. Kabel ini berfungsi ganda:
- Memberikan suplai daya 5V ke board Arduino.
 - Mengirimkan data pembacaan suhu via komunikasi serial ke Serial Monitor laptop (di port COM3, baud rate 9600).
4. Program Arduino

```
#define LM35_PIN A1
#define FAN_PIN 3
#define THRESHOLD 35.0

void setup() {
  pinMode(FAN_PIN, OUTPUT);
  digitalWrite(FAN_PIN, LOW);
  Serial.begin(9600);

  Serial.println("=====");
  Serial.println(" MONITOR SUHU LM35");
  Serial.println(" Threshold Kipas = 35 C");
  Serial.println("=====");
}

void loop() {
  int nilaiADC = analogRead(LM35_PIN);
  float tegangan = nilaiADC * (5.0 / 1023.0);
  float suhu = tegangan * 100.0;
  Serial.print("ADC    : ");
```

```

Serial.println(nilaiADC);
Serial.print("Tegangan : ");
Serial.print(tegangan, 3);
Serial.println(" V");
Serial.print("Suhu   : ");
Serial.print(suhu, 2);
Serial.println(" C");

// Kontrol kipas berdasarkan threshold
if (suhu >= THRESHOLD) {
    digitalWrite(FAN_PIN, HIGH);
    Serial.println("Status : KIPAS ON");
} else {
    digitalWrite(FAN_PIN, LOW);
    Serial.println("Status : KIPAS OFF");
    Serial.println("Kondisi : SUHU < 35 C");
}
Serial.println("-----");
delay(1000);
}

```

Program Arduino ini dirancang untuk membaca suhu menggunakan sensor LM35 dan mengendalikan motor kipas berdasarkan nilai suhu yang terukur. Pada bagian awal program digunakan `#define LM35_PIN A1` untuk menentukan bahwa keluaran sensor LM35 terhubung ke pin analog A1, sedangkan `#define FAN_PIN 3` menentukan bahwa kipas dikendalikan melalui pin digital D3. Selanjutnya, `#define THRESHOLD 35.0` menetapkan nilai batas suhu sebesar 35°C. Dengan demikian, nilai suhu 35°C menjadi acuan Arduino untuk menentukan apakah kipas harus diaktifkan atau dimatikan.

Fungsi `setup()` digunakan untuk melakukan proses inisialisasi yang hanya dijalankan satu kali ketika Arduino mulai bekerja. Perintah `pinMode(FAN_PIN, OUTPUT)` menetapkan pin D3 sebagai pin keluaran karena pin tersebut digunakan untuk memberikan sinyal kontrol kepada kipas. Perintah `digitalWrite(FAN_PIN, LOW)` memastikan kipas berada dalam kondisi mati pada saat awal sistem dijalankan. Selanjutnya, `Serial.begin(9600)` mengaktifkan komunikasi serial antara Arduino dan komputer dengan kecepatan 9600 baud. Beberapa perintah `Serial.println()` digunakan untuk

menampilkan informasi awal pada Serial Monitor, seperti judul sistem dan nilai threshold yang digunakan.

Setelah proses inisialisasi selesai, Arduino menjalankan fungsi *loop()* secara berulang. Baris *int nilaiADC = analogRead(LM35_PIN)* digunakan untuk membaca tegangan analog dari sensor LM35 melalui pin A1. Arduino Uno memiliki ADC dengan resolusi 10-bit, sehingga hasil pembacaan berada pada rentang 0 sampai 1023. Nilai ADC tersebut kemudian dikonversi menjadi tegangan menggunakan persamaan *float tegangan = nilaiADC * (5.0 / 1023.0)*. Persamaan tersebut menggunakan tegangan referensi 5 V Arduino untuk mendapatkan nilai tegangan yang dihasilkan oleh sensor LM35.

Setelah tegangan diperoleh, program menghitung nilai suhu menggunakan perintah *float suhu = tegangan * 100.0*. Sensor LM35 memiliki karakteristik keluaran sekitar 10 mV untuk setiap kenaikan suhu 1°C. Oleh karena itu, tegangan keluaran sensor dikalikan 100 untuk memperoleh nilai suhu dalam satuan Celsius. Sebagai contoh, apabila tegangan yang terbaca adalah 0,35 V, maka suhu yang dihitung adalah sekitar 35°C. Hasil pembacaan ADC, tegangan, dan suhu kemudian ditampilkan pada Serial Monitor menggunakan *Serial.print()* dan *Serial.println()* sehingga pengguna dapat memantau kondisi sistem secara *real-time*.

Bagian berikutnya merupakan proses pengambilan keputusan menggunakan struktur *if...else*. Perintah *if (suhu >= THRESHOLD)* digunakan untuk membandingkan suhu yang telah dihitung dengan nilai threshold 35°C. Apabila suhu lebih besar atau sama dengan 35°C, Arduino menjalankan *digitalWrite(FAN_PIN, HIGH)* sehingga pin D3 berada pada kondisi HIGH dan kipas diperintahkan untuk ON. Program juga menampilkan informasi "Status : KIPAS ON" pada Serial Monitor. Sebaliknya, apabila suhu masih berada di bawah 35°C, kondisi *else* dijalankan dan perintah *digitalWrite(FAN_PIN, LOW)* membuat kipas berada dalam kondisi OFF.

Pada bagian akhir *loop()*, perintah *Serial.println("-----")* digunakan sebagai garis pemisah agar hasil pembacaan pada Serial Monitor lebih mudah dibaca. Selanjutnya, *delay(1000)* memberikan waktu tunggu selama 1000 milidetik atau 1 detik. Setelah waktu tunggu selesai, Arduino kembali ke awal fungsi *loop()*, membaca sensor LM35 kembali, menghitung suhu,

membandingkannya dengan threshold, kemudian mengendalikan kipas. Dengan mekanisme tersebut, sistem bekerja secara terus-menerus sebagai sistem kontrol suhu sederhana berbasis Arduino, di mana LM35 berfungsi sebagai sensor, Arduino sebagai pengolah data dan pengambil keputusan, kipas sebagai aktuator, serta Serial Monitor sebagai media pemantauan data.

BAB 5

INTERFACING PERANGKAT KERAS

Bab ini menjelaskan tentang :

- Komunikasi dengan perangkat eksternal
- Menghubungkan display dan aktuator

Interfacing perangkat keras merupakan proses atau metode untuk menghubungkan unit pemroses utama, seperti mikroprosesor, komputer, atau mikrokontroler, dengan komponen fisik eksternal agar keduanya dapat saling berkomunikasi dan bertukar data secara akurat. Jembatan komunikasi ini sangat penting karena dunia nyata bekerja dengan berbagai besaran fisik analog, sedangkan unit pemroses hanya dapat membaca dan memproses sinyal digital biner. Melalui interfacing yang tepat, sistem komputasi mampu menerima masukan kondisi lingkungan nyata dari sensor, mengolahnya sesuai program yang dirancang, lalu mengirimkan sinyal kendali ke aktuator fisik.

Untuk mewujudkan komunikasi yang stabil, sistem antarmuka ini mengandalkan beberapa elemen penting. Komponen tersebut meliputi jalur transmisi fisik berupa port I/O dan bus data, rangkaian pengondisi sinyal untuk menyamakan level tegangan dan memfilter derau, serta konverter sinyal seperti ADC (*Analog-to-Digital Converter*) dan DAC (*Digital-to-Analog Converter*). Di sisi perangkat lunak, program *driver* tingkat rendah bertindak sebagai pengatur protokol komunikasi, sinkronisasi waktu, dan tata kelola alamat agar data yang dikirim tidak mengalami tabrakan atau korupsi sinyal selama proses transfer.

Metode pengiriman datanya sendiri terbagi menjadi dua jalur utama, yaitu transmisi serial dan paralel. Komunikasi serial mengirimkan data secara berurutan bit per bit melalui satu saluran transmisi, seperti protokol UART, SPI, atau I2C, yang efisien dalam penggunaan kabel dan stabil untuk jarak menengah hingga jauh. Sebaliknya, komunikasi paralel mentransfer banyak bit sekaligus secara bersamaan melalui beberapa jalur konduktor terpisah, menawarkan kecepatan tinggi untuk komunikasi internal berjarak sangat dekat.

Penerapannya terlihat nyata pada otomatisasi industri, sistem robotika, otomotif, hingga perangkat *smart home*. Salah satu contohnya

adalah sistem penyiram tanaman otomatis, di mana mikrokontroler membaca data kelembapan tanah melalui sensor lewat pin ADC, memproses ambang batas kekeringannya, lalu mengirim instruksi digital ke modul relai untuk menyalakan pompa air secara mandiri tanpa campur tangan manusia. Integrasi antarmuka yang solid memastikan perangkat komputasi dapat memantau dan memanipulasi dunia fisik secara responsif dan presisi.

5.1 Komunikasi dengan perangkat eksternal

Komunikasi antara mikrokontroler Arduino dan perangkat eksternal (seperti sensor, modul komunikasi, aktuator, dan memori) merupakan inti dari perancangan sistem tertanam. Proses ini membutuhkan integrasi yang tepat antara parameter kelistrikan, pemilihan protokol transmisi data, serta struktur perangkat lunak yang andal.

5.1.1 Karakteristik Elektrikal dan Level Tegangan

Sebelum menghubungkan perangkat, perhatikan batas fisik dan tegangan operasional:

- **Tegangan Logika (5V vs 3.3V)** : Arduino klasik (seperti Uno atau Nano) beroperasi pada level logika 5V, sedangkan modul modern (ESP32, modul SD card, sensor presisi) banyak menggunakan 3.3V. Menghubungkan output 5V langsung ke pin 3.3V dapat merusak modul. Untuk itu, diperlukan modul *logic level shifter* (dua arah untuk I2C atau satu arah untuk UART/SPI).
- **Batas Arus Pin** : Pin digital Arduino umumnya memiliki batas arus aman sekitar 20 mA. Beban berat seperti motor, relai mekanik, atau lampu daya tinggi tidak boleh disuplai langsung dari pin Arduino, melainkan harus menggunakan transistor, MOSFET, atau modul penggerak (*driver*) tersendiri.

5.1.2 Protokol Komunikasi Utama

1) *UART (Universal Asynchronous Receiver-Transmitter)*

Karakteristik : Komunikasi serial asinkron (tanpa sinyal *clock* bersama). Menggunakan dua jalur data utama: **TX** (kirim) dan **RX** (terima), dengan konfigurasi pin silang (TX Arduino ke RX perangkat, dan RX Arduino ke TX perangkat).

Fitur Utama : Memerlukan kesepakatan laju transmisi (*baud rate*, misalnya 9600 atau 115200 bps) dan format bingkai data yang sama (data bit, paritas, *stop bit*).

Penerapan : Komunikasi Arduino ke komputer lewat kabel USB, modul GPS, Bluetooth (HC-05), dan modem GSM. Untuk komunikasi jarak jauh di lingkungan industri, UART dapat dikonversi ke sinyal diferensial melalui modul **RS-485**.

2) SPI (Serial Peripheral Interface)

Karakteristik : Komunikasi serial sinkron berkecepatan sangat tinggi, bersifat *full-duplex* (kirim dan terima simultan), menggunakan topologi Master-Slave.

Jalur Fisik (4 Jalur):

- **SCK :** Jalur pulsa waktu (*clock*) yang dikendalikan oleh Arduino (Master).
- **MOSI :** Jalur pengiriman data dari Master ke Slave.
- **MISO :** Jalur pengiriman data dari Slave ke Master.
- **SS / CS :** Jalur pemilihan modul target. Untuk menghubungkan beberapa perangkat Slave, Arduino menggunakan pin CS terpisah untuk tiap modul.

Penerapan : Modul pembaca kartu microSD, tampilan grafis TFT LCD, modul Ethernet W5500, dan sensor berkecepatan tinggi lainnya pada jarak kabel yang pendek.

3) I2C (Inter-Integrated Circuit)

Karakteristik : Komunikasi serial sinkron hemat jalur dengan topologi bus bersama, memungkinkan banyak modul terhubung hanya menggunakan dua kabel.

Jalur Fisik (2 Jalur):

- **SDA :** Jalur transfer data serial dua arah.
- **SCL :** Jalur pulsa *clock* pengatur sinkronisasi.

Fitur Utama : Menggunakan sistem pengalamatan biner unik (alamat 7-bit) untuk setiap modul, arsitektur *open-drain* yang

mewajibkan adanya resistor *pull-up*, serta sinyal konfirmasi (*ACK/NACK*).

Penerapan : Sensor lingkungan (BME280, DHT12), sensor gerak/IMU (MPU-6050), modul RTC (DS3231), dan layar karakter LCD dengan modul backpack I2C.

5.1.3 Sinyal Kontrol Khusus

- 1) **PWM (Pulse Width Modulation)** : Mengontrol daya rata-rata keluaran digital secara semu dengan mengatur lebar pulsa aktif (*duty cycle*), umum dipakai untuk redup-terang lampu LED dan pengatur kecepatan motor. Sinyal pulsa dengan durasi tertentu juga digunakan untuk mengendalikan sudut poros servomotor.
- 2) **Interupsi Eksternal (Hardware Interrupts)** : Menangani sinyal masukan mendesak secara instan tanpa menunggu alur program utama berjalan (misalnya membaca pulsa putaran enkoder motor cepat atau tombol darurat).
- 3) **1-Wire** : Protokol serial jalur tunggal (satu kawat data plus referensi *ground*), seperti pada sensor suhu digital DS18B20 yang memungkinkan banyak sensor diparalelkan pada satu kabel.

5.1.4 Integritas Sinyal dan Keandalan Sistem

- 1) **Pencegahan Gangguan Listrik** : Menghubungkan ground Arduino dan ground modul eksternal secara bersamaan (*common ground*) adalah syarat wajib agar referensi tegangan setara. Jika modul menggunakan catu daya tegangan tinggi terpisah, gunakan isolator optokopler (*optocoupler*) untuk melindungi pin mikrokontroler.
- 2) **Kapasitor Bypass / Decoupling**: Memasang kapasitor kecil sedekat mungkin dengan pin daya modul eksternal untuk menyaring lonjakan tegangan transien.
- 3) **Verifikasi Data** : Pada pengiriman data nirkabel atau kabel panjang, sertakan metode pengecekan data seperti *Checksum* atau *CRC* untuk memastikan data yang diterima tidak mengalami kerusakan akibat derau (*noise*) elektromagnetik.

5.2 Menghubungkan display dan aktuator

Menghubungkan layar tampilan (**LCD dengan modul I2C**) dan aktuator (**Motor Servo**) ke mikrokontroler arduino adalah salah satu kombinasi proyek antarmuka (*interfacing*) yang paling umum. Proyek ini menggabungkan dua metode komunikasi sekaligus:

- ✓ **Komunikasi Bus I2C** untuk mengirim teks ke LCD secara efisien (hanya butuh 2 pin data).
- ✓ **Sinyal PWM (Pulse Width Modulation)** untuk mengontrol sudut pergerakan motor servo (1 pin sinyal kontrol).

Berikut penjelasan detail mulai dari prinsip kerja, kebutuhan komponen, skema pengkabelan, hingga implementasi kodenya.

1. Prinsip Kerja Masing-Masing Komponen

a. **LCD I2C (Liquid Crystal Display + PCF8574 Backpack) :**

Layar LCD karakter standar (seperti tipe 16x2 atau 20x4) aslinya membutuhkan setidaknya 6 hingga 10 pin mikrokontroler jika dipasang secara paralel. Dengan menambahkan modul *backpack* I2C (berbasis chip konverter I/O PCF8574), mikrokontroler cukup mengirim perintah lewat dua jalur serial: **SDA** (*Serial Data*) dan **SCL** (*Serial Clock*). Setiap modul memiliki alamat I2C unik (biasanya **0x27** atau **0x3F**). Di bagian belakang modul terdapat potensiometer kecil (*trimmer*) untuk mengatur kontras ketajaman huruf.

b. **Motor Servo (misalnya TowerPro SG90 / MG996R) :**

Motor servo adalah aktuator putar tertutup (*closed-loop*) yang dilengkapi motor DC kecil, roda gigi reduksi, potensiometer internal sebagai sensor posisi, dan sirkuit kendali.

Servo tidak sekadar diberi tegangan on/off, melainkan dikendalikan oleh **durasi pulsa PWM** yang diulang setiap 20 milidetik (frekuensi 50 Hz):

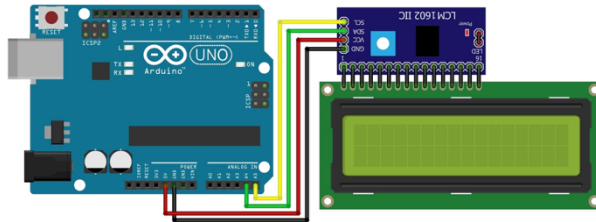
- ✓ Pulsa sekitar 1,0 milidetik → bergerak ke sudut 0°.
- ✓ Pulsa sekitar 1,5 milidetik → bergerak ke sudut netral 90°.
- ✓ Pulsa sekitar 2,0 milidetik → bergerak ke sudut maksimum 180°.

Catatan Penting : Motor servo saat mulai berputar atau menahan beban menarik lonjakan arus yang cukup besar (bisa mencapai 500 mA – 1 A). Mengambil daya servo langsung dari pin 5V Arduino

dapat memicu penurunan tegangan (*voltage drop*) yang menyebabkan Arduino mereset berulang kali atau merusak regulator onboard.

2. Diagram Pengkabelan (*Wiring Connection*)

A. Koneksi LCD I2C ke Arduino

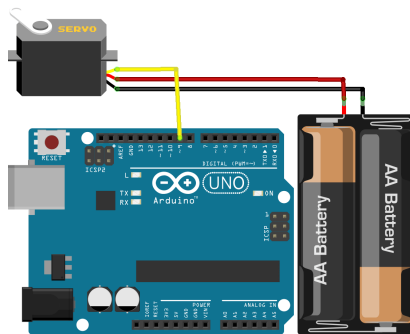


Gambar 21. Wiring LCD i2c

Modul I2C memiliki 4 pin:

- ✓ **GND** → Pin **GND** Arduino.
- ✓ **VCC** → Pin **5V** Arduino.
- ✓ **SDA** → Pin **A4** (Arduino Uno) atau pin khusus **SDA** di sudut kiri atas board.
- ✓ **SCL** → Pin **A5** (Arduino Uno) atau pin khusus **SCL** di sudut kiri atas board.

B. Koneksi Motor Servo



Gambar 22. Wiring Servo

Motor servo standar memiliki kabel 3 warna:

- ✓ **Cokelat / Hitam (Ground):** Hubungkan ke **GND** Catu Daya Eksternal (dan **wajib dihubungkan ke GND Arduino / Common Ground**).
- ✓ **Merah (VCC / Daya):** Hubungkan ke kutub **Positif (+5V)** Catu Daya Eksternal. *(Jika hanya uji coba tanpa beban, bisa sementara ke pin 5V Arduino).*
- ✓ **Oranye / Kuning (Sinyal PWM):** Hubungkan ke salah satu pin digital PWM Arduino (misalnya **Pin D9**).

Aturan Wajib (Common Ground): Jika menggunakan catu daya eksternal untuk servo, kutub negatif/GND dari catu daya eksternal **harus disatukan** dengan pin GND Arduino agar referensi level tegangan sinyal kendali PWM seimbang dan terbaca sempurna oleh servo.

3. Pustaka (*Libraries*) yang Diperlukan

Pada Arduino IDE, pastikan dua pustaka berikut terpasang:

- ✓ Wire.h: Pustaka bawaan Arduino untuk protokol komunikasi I2C.
- ✓ Servo.h: Pustaka bawaan Arduino untuk membangkitkan sinyal kontrol servo.
- ✓ LiquidCrystal_I2C.h: Pustaka eksternal untuk mengontrol modul LCD I2C (bisa diunduh via Library Manager Arduino IDE dengan mengetik LiquidCrystal I2C oleh Frank de Brabander / Marco Schwartz).

4. Kode Program Contoh

Kode berikut menggerakkan servo secara bertahap dari 0° ke 180° lalu menampilkannya secara real-time pada layar LCD 16x2.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
Servo myServo;
const int SERVO_PIN = 9;

void setup() {
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0); // Kolom 0, Baris 0
```

```

    lcd.print("Sistem Aktuator");
    lcd.setCursor(0, 1); // Kolom 0, Baris 1
    lcd.print("Inisialisasi...");
    delay(1500);
    lcd.clear();
    myServo.attach(SERVO_PIN);
    myServo.write(0);
    delay(500);
}

void loop() {
    // Putar dari 0 derajat ke 180 derajat
    for (int pos = 0; pos <= 180; pos += 30) {
        myServo.write(pos);      // Gerakkan servo ke sudut 'pos'
        updateLCD(pos);          // Tampilkan informasi ke layar LCD
        delay(500);              // Beri jeda waktu mekanik bergerak
    }
    delay(1000);
    for (int pos = 180; pos >= 0; pos -= 30) {
        myServo.write(pos);
        updateLCD(pos);
        delay(500);
    }
    delay(1000);
}

void updateLCD(int sudut) {
    lcd.setCursor(0, 0);
    lcd.print("Status: BERGERAK ");
    lcd.setCursor(0, 1);
    lcd.print("Sudut: ");
    lcd.print(sudut);
    lcd.print((char)223);
    lcd.print(" ");
}

```

BAB 6

DESAIN PERANGKAT LUNAK TERSEMAT (*EMBEDEED SOFTWARE*)

Bab ini menjelaskan tentang :

- o Menulis kode yang efisien untuk mikrokontroler
- o Interupsi, timer dan kontrol real-time
- o Debugging dan optimasi kinerja

Perangkat lunak *embedded* merupakan bagian penting dalam sistem berbasis mikrokontroler karena berfungsi mengendalikan perangkat keras agar dapat menjalankan tugas sesuai kebutuhan sistem. Berbeda dengan perangkat lunak pada komputer umum yang memiliki sumber daya relatif besar, perangkat lunak *embedded* bekerja dengan keterbatasan memori, kapasitas pemrosesan, konsumsi daya, dan waktu eksekusi. Oleh karena itu, perancangan perangkat lunak *embedded* harus mempertimbangkan efisiensi, ketepatan waktu, keandalan, serta kemampuan sistem dalam merespons perubahan kondisi secara cepat.

Pada sistem mikrokontroler, perangkat lunak umumnya terdiri atas program utama (*main program*), fungsi-fungsi pendukung, pembacaan sensor, pengolahan data, serta pengendalian aktuator. Struktur program yang baik perlu dibuat modular sehingga setiap fungsi memiliki tugas yang jelas. Pendekatan modular mempermudah proses pengembangan, pengujian, pemeliharaan, dan pengembangan sistem pada tahap berikutnya.

6.1 Menulis kode yang efisien untuk mikrokontroler

Efisiensi perangkat lunak merupakan salah satu aspek penting dalam perancangan sistem *embedded*. Mikrokontroler memiliki sumber daya yang jauh lebih terbatas dibandingkan komputer, baik dari sisi kapasitas memori, kecepatan prosesor, konsumsi daya, maupun jumlah periferal yang tersedia. Oleh karena itu, program yang dikembangkan untuk mikrokontroler harus dirancang agar mampu menyelesaikan tugas dengan penggunaan sumber daya yang optimal.

Kode yang efisien bukan hanya kode yang memiliki jumlah baris sedikit. Program yang efisien harus mampu menggunakan memori secara tepat, memiliki waktu eksekusi yang sesuai dengan kebutuhan aplikasi, tidak melakukan proses yang tidak diperlukan, serta tetap mudah dibaca dan dipelihara. Dalam aplikasi industri, efisiensi juga berkaitan dengan kemampuan sistem merespons perubahan input secara konsisten dan tepat waktu.

6.1.1 Memahami Keterbatasan Sumber Daya

Sebelum menulis program, programmer perlu memahami karakteristik mikrokontroler yang digunakan. Sebagai contoh, Arduino Uno berbasis ATmega328P memiliki Flash sebesar 32 KB, SRAM sebesar 2 KB, dan EEPROM sebesar 1 KB. Kapasitas tersebut cukup untuk berbagai aplikasi otomasi sederhana, tetapi dapat menjadi keterbatasan ketika program menggunakan banyak variabel, *string*, pustaka, atau algoritma yang kompleks.

Memori pada mikrokontroler secara umum dapat dibedakan menjadi beberapa bagian. Flash digunakan untuk menyimpan program, SRAM digunakan selama program berjalan, sedangkan EEPROM dapat digunakan untuk menyimpan data yang harus tetap tersedia ketika mikrokontroler kehilangan catu daya.

Kesalahan dalam penggunaan SRAM dapat menyebabkan program mengalami perilaku yang tidak terduga. Oleh karena itu, programmer perlu mempertimbangkan ukuran setiap variabel. Jika suatu nilai hanya membutuhkan rentang 0–255, penggunaan tipe data `byte` dapat lebih hemat dibandingkan tipe data yang lebih besar.

Contoh:

```
byte statusMotor = 1;  
int nilaiSensor = 0;  
float suhu = 0.0;
```

Pemilihan tipe data tersebut harus tetap mempertimbangkan kebutuhan perhitungan. Penghematan memori tidak boleh mengorbankan ketepatan data.

6.1.2 Menghindari Penggunaan `delay()` Secara Berlebihan

Salah satu kesalahan yang sering ditemukan pada program mikrokontroler pemula adalah penggunaan `delay()` secara berlebihan. Fungsi `delay()` menghentikan eksekusi program selama waktu tertentu. Perhatikan contoh berikut :

```
digitalWrite(LED_BUILTIN, HIGH);  
delay(1000);
```

```
digitalWrite(LED_BUILTIN, LOW);  
delay(1000);
```

Program tersebut akan menyalakan LED selama satu detik dan mematikannya selama satu detik. Namun, selama `delay()` berlangsung, mikrokontroler tidak dapat menjalankan bagian program utama lainnya secara normal.

Masalah tersebut menjadi lebih jelas ketika mikrokontroler harus melakukan beberapa pekerjaan secara bersamaan. Misalnya, sistem harus membaca sensor suhu, memantau tombol, mengendalikan motor, dan mengirimkan data melalui komunikasi serial. Jika program menggunakan `delay(5000)`, sistem dapat kehilangan kesempatan untuk merespons input selama lima detik. Alternatif yang lebih baik adalah menggunakan pendekatan berbasis waktu dengan `millis()`. Perhatikan contoh berikut :

```
unsigned long waktuSebelumnya = 0;
const unsigned long interval = 1000;

void loop() {
    unsigned long waktuSekarang = millis();
    if (waktuSekarang - waktuSebelumnya >= interval) {
        waktuSebelumnya = waktuSekarang;
        digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
    }

    // Proses lain tetap dapat dijalankan

}
```

Dengan pendekatan tersebut, mikrokontroler tidak perlu berhenti menunggu. Program dapat melakukan tugas lain sambil tetap menjalankan proses berkala.

6.1.3 Mengurangi Proses yang Tidak Diperlukan

Efisiensi juga dapat dicapai dengan memastikan setiap proses hanya dilakukan ketika diperlukan. Sebagai contoh, jika sensor suhu hanya perlu dibaca setiap satu detik, tidak diperlukan pembacaan ribuan kali dalam satu detik.

Program yang tidak efisien:

```
void loop() {  
    int sensor = analogRead(A0);  
    float suhu = sensor * 0.488;  
    Serial.println(suhu);  
}
```

Program tersebut akan membaca sensor dan mengirim data serial berulang-ulang selama `loop()` berjalan. Pada Arduino Uno, proses tersebut dapat berlangsung sangat sering.

Jika kebutuhan sistem hanya satu pembacaan per detik, program dapat dirancang sebagai berikut:

```
unsigned long previousMillis = 0;  
const unsigned long interval = 1000;  
  
void loop() {  
    if (millis() - previousMillis >= interval) {  
        previousMillis = millis();  
        int sensor = analogRead(A0);  
        float suhu = sensor * 0.488;  
        Serial.println(suhu);  
    }  
}
```

Pendekatan ini mengurangi jumlah pembacaan sensor dan komunikasi serial sehingga beban prosesor menjadi lebih rendah.

6.1.4 Menggunakan Struktur Program yang Modular

Program yang efisien sebaiknya tidak ditulis sebagai satu blok kode yang panjang. Program dapat dibagi menjadi beberapa fungsi berdasarkan tugasnya.

Sebagai contoh, sistem pengontrol kipas berdasarkan sensor suhu dapat memiliki fungsi:

```
void bacaSensor();  
void kendalikanKipas();  
void tampilkanData();
```

Kemudian program utama dapat ditulis:

```
void loop() {  
    bacaSensor();  
    kendalikanKipas();  
    tampilkanData();  
}
```

Struktur tersebut membuat alur program lebih mudah dipahami. Apabila terjadi kesalahan pada pembacaan sensor, programmer dapat memeriksa fungsi `bacaSensor()` tanpa harus menganalisis seluruh program.

Modularitas juga mempermudah penggunaan kembali kode (*code reuse*). Fungsi yang telah dibuat untuk membaca sensor tertentu dapat digunakan kembali pada proyek lain dengan sedikit perubahan.

6.1.5 Mengoptimalkan Penggunaan Variabel

Variabel merupakan bagian penting dalam program mikrokontroler. Namun, penggunaan variabel yang tidak diperlukan dapat meningkatkan penggunaan memori.

Sebagai contoh:

```
int status = 0;  
int statusMotor = 0;  
int statusRelay = 0;
```

Jika variabel tersebut hanya menyimpan nilai 0 atau 1, penggunaan tipe data yang lebih kecil dapat dipertimbangkan:

```
byte statusMotor = 0;  
byte statusRelay = 0;
```

Pada sistem dengan banyak variabel, penghematan kecil pada setiap variabel dapat memberikan pengaruh terhadap penggunaan SRAM secara keseluruhan.

Programmer juga perlu menghindari pembuatan variabel global secara berlebihan. Variabel global memang mudah digunakan, tetapi terlalu banyak variabel global dapat membuat struktur program sulit dipahami dan meningkatkan kemungkinan terjadinya perubahan data yang tidak disengaja.

6.1.6 Menghindari Penggunaan *String* yang Tidak Terkontrol

Pada mikrokontroler dengan SRAM terbatas, penggunaan objek *String* secara berlebihan perlu diperhatikan. Operasi *string* yang sering membuat dan menghapus objek dapat menyebabkan fragmentasi memori. Sebagai alternatif, untuk aplikasi sederhana dapat digunakan array karakter :

```
char pesan[] = "Motor ON";
```

Untuk teks yang bersifat konstan dan tidak perlu dimodifikasi, penyimpanan pada Flash juga dapat dipertimbangkan menggunakan mekanisme yang sesuai dengan platform mikrokontroler.

Teknik tersebut menjadi penting pada aplikasi yang berjalan dalam waktu lama, misalnya sistem monitoring energi atau sistem IoT yang diharapkan bekerja selama berhari-hari atau berbulan-bulan tanpa *restart*.

6.1.7 Mengurangi Operasi Perhitungan yang Berulang

Operasi matematika yang sama tidak perlu dilakukan berulang-ulang apabila hasilnya tidak berubah. Sebagai contoh:

```
void loop() {  
    float faktor = 5.0 / 1023.0;  
    int sensor = analogRead(A0);  
    float tegangan = sensor * faktor;  
}
```

Nilai 5.0 / 1023.0 tidak berubah selama tegangan referensi dan resolusi ADC tetap sama. Oleh karena itu, faktor tersebut dapat dihitung sekali:

```
const float faktorADC = 5.0 / 1023.0;
void loop() {
    int sensor = analogRead(A0);
    float tegangan = sensor * faktorADC;
}
```

Pendekatan tersebut membuat program lebih terstruktur dan menghindari perhitungan yang tidak diperlukan.

6.1.8 Contoh Penerapan: Sistem Kontrol Suhu dengan Arduino

Sebagai contoh penerapan, sebuah sistem menggunakan Arduino Uno dan sensor LM35 untuk mengukur suhu. Kipas akan aktif apabila suhu mencapai atau melebihi 35 °C dan mati ketika suhu berada di bawah nilai tersebut.

Program sederhana dapat dirancang sebagai berikut:

```
const int pinLM35 = A0;
const int pinKipas = 8;
const float batasSuhu = 35.0;
unsigned long waktuSebelumnya = 0;
const unsigned long interval = 500;

void setup() {
    pinMode(pinKipas, OUTPUT);
    digitalWrite(pinKipas, LOW);
    Serial.begin(9600);
}

void loop() {
    unsigned long waktuSekarang = millis();
    if (waktuSekarang - waktuSebelumnya >= interval) {
        waktuSebelumnya = waktuSekarang;
        int nilaiADC = analogRead(pinLM35);
        float tegangan = nilaiADC * (5.0 / 1023.0);
    }
```

```

float suhu = tegangan * 100.0;
if (suhu >= batasSuhu) {
    digitalWrite(pinKipas, HIGH);
} else {
    digitalWrite(pinKipas, LOW);
}
Serial.print("Suhu: ");
Serial.print(suhu);
Serial.println(" C");
}
}

```

Program tersebut menerapkan beberapa prinsip efisiensi. Pertama, pembacaan sensor dilakukan setiap 500 ms sehingga tidak dilakukan terus-menerus. Kedua, `millis()` digunakan sebagai pengatur waktu sehingga program tidak berhenti seperti ketika menggunakan `delay()`. Ketiga, nilai batas suhu disimpan sebagai konstanta sehingga mudah diubah tanpa harus mencari angka tersebut di berbagai bagian program.

Jika sistem dikembangkan lebih lanjut, program dapat ditambahkan histeresis untuk mencegah kipas terlalu sering hidup dan mati ketika suhu berada di sekitar 35 °C. Misalnya, kipas dinyalakan pada suhu ≥ 35 °C dan baru dimatikan ketika suhu turun hingga ≤ 33 °C.

```

if (suhu >= 35.0) {
    digitalWrite(pinKipas, HIGH);
}

if (suhu <= 33.0) {
    digitalWrite(pinKipas, LOW);
}

```

Pendekatan tersebut merupakan contoh bagaimana efisiensi perangkat lunak tidak hanya berkaitan dengan penggunaan memori atau kecepatan eksekusi, tetapi juga berkaitan dengan perancangan algoritma yang menghasilkan perilaku sistem yang stabil.

6.2 Interupsi, timer dan kontrol *real-time*

Pada sistem berbasis mikrokontroler, kemampuan untuk merespons suatu kejadian dalam waktu yang tepat merupakan salah satu karakteristik utama sistem *embedded*. Mikrokontroler tidak hanya dituntut menghasilkan keluaran yang benar, tetapi juga harus menghasilkan keluaran tersebut pada waktu yang sesuai. Sebuah sistem pengendali motor, misalnya, harus mampu membaca sensor, memproses data, dan memberikan perintah kepada aktuator dalam interval waktu tertentu. Apabila respons terlalu lambat atau tidak konsisten, kinerja sistem dapat menurun bahkan menyebabkan kegagalan proses.

Untuk memenuhi kebutuhan tersebut, mikrokontroler menyediakan beberapa mekanisme penting, yaitu **interupsi** (*interrupt*), **timer**, dan mekanisme pengendalian **real-time**. Ketiganya saling berkaitan dalam membangun perangkat lunak yang responsif dan memiliki perilaku waktu yang terprediksi.

6.2.1 Konsep Interupsi

Interupsi adalah mekanisme yang memungkinkan mikrokontroler menghentikan sementara proses utama ketika terjadi suatu kejadian tertentu. Setelah kejadian tersebut ditangani, mikrokontroler dapat kembali melanjutkan proses utama.

Tanpa interupsi, program utama harus terus memeriksa kondisi berbagai perangkat menggunakan metode *polling*. Misalnya, untuk mengetahui apakah sebuah tombol ditekan, program dapat melakukan:

```
if (digitalRead(tombol) == LOW) {  
    // tombol ditekan  
}
```

Metode tersebut disebut *polling* karena mikrokontroler secara aktif memeriksa kondisi input. Jika terdapat banyak proses dalam program, pemeriksaan tombol mungkin terlambat dilakukan.

Interupsi memberikan pendekatan yang berbeda. Mikrokontroler dapat menjalankan program utama dan hanya menjalankan fungsi tertentu ketika kejadian yang ditentukan terjadi.

Pada Arduino Uno, salah satu penggunaan umum adalah interupsi eksternal pada pin digital 2 dan 3. Contohnya adalah:

```
attachInterrupt(digitalPinToInterrupt(2), tombolDitekan, FALLING);
```

Ketika terjadi perubahan kondisi pada pin tersebut sesuai mode yang dipilih, mikrokontroler akan menjalankan fungsi

6.2.2 Interrupt Service Routine

Fungsi yang dijalankan ketika terjadi interupsi disebut **Interrupt Service Routine (ISR)**. ISR harus dirancang dengan hati-hati karena selama ISR dijalankan, proses utama dapat terganggu.

```
volatile bool tombolAktif = false;
```

```
void tombolDitekan() {  
    tombolAktif = true;  
}
```

Variabel yang digunakan bersama antara program utama dan ISR sebaiknya diperhatikan secara khusus. Kata kunci `volatile` digunakan untuk memberi tahu compiler bahwa nilai variabel dapat berubah di luar alur eksekusi normal program.

Program utama kemudian dapat memproses kejadian tersebut:

```
void loop() {  
    if (tombolAktif) {  
        tombolAktif = false;  
        // Proses utama  
        Serial.println("Tombol ditekan");  
    }  
}
```

Pendekatan ini lebih aman dibandingkan melakukan proses panjang langsung di dalam ISR. ISR sebaiknya dibuat singkat. Hindari melakukan operasi seperti komunikasi serial yang panjang, perhitungan kompleks, atau penggunaan fungsi yang membutuhkan waktu lama. ISR idealnya hanya mencatat bahwa suatu kejadian telah terjadi, kemudian proses selanjutnya dilakukan oleh program utama.

6.2.3 Jenis Pemicu Interupsi

Interupsi digital dapat dipicu berdasarkan beberapa kondisi. Pada Arduino, mode yang umum digunakan antara lain:

- ✓ LOW: interupsi aktif ketika pin berada pada kondisi LOW.
- ✓ CHANGE: interupsi aktif ketika terjadi perubahan kondisi.
- ✓ RISING: interupsi aktif ketika sinyal berubah dari LOW ke HIGH.
- ✓ FALLING: interupsi aktif ketika sinyal berubah dari HIGH ke LOW.

Pemilihan mode bergantung pada karakteristik sensor atau perangkat input.

Sebagai contoh, sensor *encoder* menghasilkan pulsa ketika poros motor berputar. Setiap pulsa dapat digunakan sebagai sumber interupsi. Dengan menghitung jumlah pulsa dalam interval tertentu, mikrokontroler dapat menghitung kecepatan putaran motor.

6.2.4 Timer pada Mikrokontroler

Timer merupakan periferal perangkat keras yang digunakan untuk menghitung waktu atau menghasilkan kejadian secara periodik. Timer bekerja berdasarkan sinyal *clock* mikrokontroler.

Pada Arduino Uno berbasis ATmega328P terdapat beberapa timer yang dapat digunakan untuk berbagai keperluan. Timer dapat dimanfaatkan untuk menghasilkan PWM, menentukan interval waktu, menghasilkan interupsi periodik, atau membantu pengukuran waktu.

Salah satu contoh penggunaan timer secara tidak langsung adalah fungsi `millis()`. Fungsi tersebut memanfaatkan mekanisme timer internal untuk menyediakan informasi mengenai waktu sejak mikrokontroler mulai bekerja.

Contoh:

```
unsigned long sebelumnya = 0;
void loop() {
    if (millis() - sebelumnya >= 1000) {
        sebelumnya = millis();
        Serial.println("Satu detik telah berlalu");
    }
}
```

Program tersebut memungkinkan proses periodik dilakukan tanpa menghentikan eksekusi program.

6.2.5 Timer dan PWM

Timer juga digunakan untuk menghasilkan **Pulse Width Modulation (PWM)**. PWM banyak digunakan untuk mengatur kecepatan motor DC, intensitas LED, atau keluaran aktuator lainnya.

Pada Arduino Uno, fungsi:

```
analogWrite(9, 128);
```

Menghasilkan sinyal PWM pada pin yang mendukung PWM. Nilai 128 pada rentang 0–255 menghasilkan *duty cycle* sekitar 50%.

6.2.6 Konsep Sistem Real-Time

Sistem *real-time* adalah sistem yang harus memberikan respons dalam batas waktu tertentu. Dalam sistem real-time, kebenaran sistem tidak hanya ditentukan oleh hasil perhitungan, tetapi juga oleh kapan hasil tersebut tersedia.

Sebagai contoh, sebuah sistem kontrol suhu mungkin memiliki batas waktu pembacaan sensor setiap 100 ms. Jika sensor baru dibaca setiap 5 detik, algoritma kontrol mungkin tidak mampu merespons perubahan suhu dengan cepat.

Sistem real-time dapat dibedakan menjadi hard real-time dan soft real-time.

Pada *hard real-time*, keterlambatan melewati batas waktu dapat menyebabkan kegagalan sistem. Contohnya adalah sistem keselamatan tertentu atau pengendalian proses yang memiliki batas waktu ketat.

Pada *soft real-time*, keterlambatan sesekali masih dapat ditoleransi, meskipun tetap memengaruhi kualitas sistem. Contohnya adalah sistem monitoring atau tampilan data.

6.2.7 Penjadwalan Tugas

Sistem mikrokontroler sering kali harus menjalankan beberapa tugas dengan periode berbeda. Sebagai contoh:

Tabel 5. Contoh penjadwalan tugas

Tugas	Periode (ms)
Membaca sensor suhu	500
Membaca sensor cahaya	100
Mengirim data IoT	5000
Memperbarui LCD	10000
Pemeriksaan tombol	10

Salah satu pendekatan sederhana adalah menggunakan *non-blocking scheduling*. Contoh:

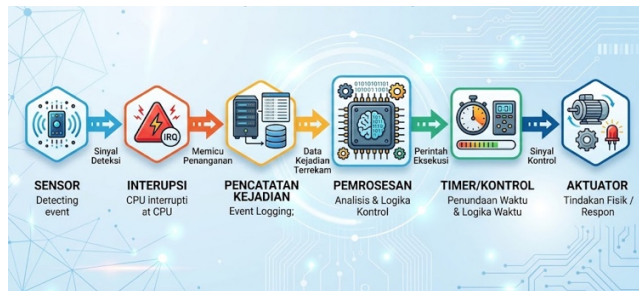
```
unsigned long waktuSensor = 0;
unsigned long waktuIoT = 0;
void loop(){
    unsigned long sekarang = millis();
    if (sekarang - waktuSensor >= 500) {
        waktuSensor = sekarang;
        bacaSensor();
    }
    if (sekarang - waktuIoT >= 5000) {
        waktuIoT = sekarang;
        kirimData();
    }
}
```

Dengan pendekatan tersebut, beberapa tugas dapat berjalan berdasarkan periode masing-masing tanpa menggunakan `delay()`.

6.2.8 Penerapan pada Sistem Kendali Otomasi

Konsep interupsi dan timer dapat dikembangkan lebih lanjut pada sistem otomasi. Misalnya pada sistem konveyor, encoder digunakan untuk mengetahui posisi atau kecepatan benda, sensor

proximity digunakan untuk mendeteksi keberadaan benda, dan timer digunakan untuk menentukan kapan aktuator harus bekerja.



Gambar 23. Alur Sistem Berbasis Event dan Kontrol

Dengan struktur tersebut, mikrokontroler tidak harus terus-menerus menunggu perubahan sensor. Sistem dapat menjalankan tugas utama dan merespons kejadian penting melalui mekanisme interupsi.

6.2.9 Prinsip Perancangan *Real-Time*

Dalam merancang perangkat lunak real-time, beberapa prinsip perlu diperhatikan:

- 1) Tentukan periode setiap tugas.
- 2) Hindari proses *blocking* yang terlalu lama.
- 3) Gunakan timer atau mekanisme penjadwalan untuk tugas periodik.
- 4) Gunakan interupsi untuk kejadian yang membutuhkan respons cepat.
- 5) Buat ISR sesingkat mungkin.
- 6) Hindari komunikasi atau perhitungan kompleks di dalam ISR.
- 7) Pastikan waktu eksekusi tugas tidak melebihi periode yang ditentukan.
- 8) Pisahkan tugas berprioritas tinggi dan rendah.
- 9) Lakukan pengujian terhadap kondisi beban maksimum.
- 10) Pastikan sistem tetap memberikan respons yang dapat diprediksi.

Dengan memahami interupsi, timer, dan kontrol real-time, programmer dapat mengembangkan perangkat lunak mikrokontroler

yang tidak hanya berfungsi secara logis, tetapi juga mampu merespons kejadian secara cepat dan terjadwal. Konsep tersebut menjadi dasar bagi pengembangan sistem otomasi, robotika, kendali motor, akuisisi data, komunikasi industri, dan berbagai aplikasi *embedded system* yang membutuhkan ketepatan waktu.

6.3 Debugging dan optimasi kinerja

Dalam pengembangan sistem *embedded*, keberhasilan suatu program tidak hanya ditentukan oleh kemampuan program menghasilkan keluaran yang benar, tetapi juga oleh kestabilan, kecepatan respons, penggunaan memori, dan keandalan sistem dalam kondisi operasi yang sebenarnya. Program yang terlihat berjalan dengan baik pada kondisi tertentu belum tentu bekerja dengan benar ketika jumlah data meningkat, sensor menghasilkan nilai yang tidak ideal, komunikasi terganggu, atau sistem beroperasi dalam waktu yang lama. Oleh karena itu, proses debugging dan optimasi kinerja merupakan bagian penting dari siklus pengembangan perangkat lunak mikrokontroler.

Debugging adalah proses sistematis untuk menemukan, menganalisis, dan memperbaiki kesalahan pada perangkat lunak. Sementara itu, optimasi kinerja merupakan proses memperbaiki program agar penggunaan sumber daya mikrokontroler menjadi lebih efisien tanpa mengubah fungsi utama sistem. Kedua proses tersebut sebaiknya dilakukan secara berurutan. Program perlu dipastikan benar terlebih dahulu, kemudian dioptimalkan.

6.3.1 Jenis Kesalahan pada Program Mikrokontroler

Kesalahan pada perangkat lunak mikrokontroler dapat dikelompokkan menjadi beberapa jenis. Pemahaman terhadap jenis kesalahan akan membantu programmer menentukan metode debugging yang tepat.

- 1) Kesalahan sintaks (*syntax error*), yaitu kesalahan penulisan aturan bahasa pemrograman. Contohnya:

```
int sensor = analogRead(A0)
```

Kode tersebut tidak memiliki tanda titik koma (;) pada akhir pernyataan. Compiler akan memberikan pesan kesalahan ketika program dikompilasi.

- 2) Kesalahan logika (*logic error*). Program berhasil dikompilasi dan dijalankan, tetapi hasilnya tidak sesuai dengan yang diharapkan. Contohnya:

```
if (suhu > 35) {  
    digitalWrite(kipas, LOW);  
}
```

Jika tujuan sistem adalah menyalakan kipas ketika suhu melebihi 35 °C, logika tersebut justru memberikan kondisi yang berlawanan.

- 3) Kesalahan waktu (*timing error*), yaitu kesalahan yang berkaitan dengan waktu eksekusi program. Contohnya, sistem membutuhkan pembacaan sensor setiap 100 ms, tetapi program menggunakan proses yang menyebabkan pembacaan baru dilakukan setiap 1 detik.
- 4) Kesalahan penggunaan sumber daya, misalnya SRAM habis, penggunaan CPU terlalu tinggi, atau program terlalu besar untuk kapasitas Flash mikrokontroler.

6.3.2 Debugging Menggunakan Serial Monitor

Salah satu teknik debugging yang paling sederhana dan banyak digunakan pada Arduino adalah komunikasi serial. Programmer dapat menampilkan nilai variabel dan status sistem melalui *Serial Monitor*.

```
int nilaiSensor;  
  
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    nilaiSensor = analogRead(A0);  
    Serial.print("Nilai ADC = ");  
    Serial.println(nilaiSensor);  
    delay(500);  
}
```

Jika sensor menghasilkan nilai yang tidak sesuai, programmer dapat melihat hasil pembacaan secara langsung. Teknik ini juga dapat digunakan untuk mengetahui tahapan eksekusi program.

```
Serial.println("Program mulai");  
Serial.println("Sensor dibaca");  
Serial.println("Motor diaktifkan");
```

Dengan demikian, programmer dapat mengetahui bagian program yang telah dieksekusi dan menemukan lokasi terjadinya masalah. Namun, penggunaan `Serial.print()` secara berlebihan perlu dihindari. Komunikasi serial membutuhkan waktu sehingga terlalu banyak pesan debugging dapat memengaruhi kinerja sistem. Setelah proses debugging selesai, pesan yang tidak diperlukan sebaiknya dihapus atau dinonaktifkan.

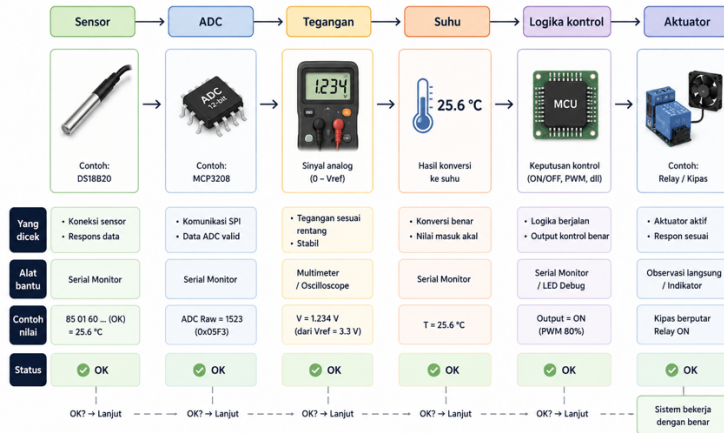
6.3.3 Debugging Berdasarkan Nilai Sensor

Dalam sistem mikrokontroler, kesalahan tidak selalu berasal dari perangkat lunak. Sensor, kabel, catu daya, atau rangkaian elektronik juga dapat menjadi sumber masalah.

Sebagai contoh, sensor LM35 dihubungkan ke ADC Arduino. Program menghitung suhu berdasarkan nilai ADC:

```
int adc = analogRead(A0);  
float tegangan = adc * (5.0 / 1023.0);  
float suhu = tegangan * 100.0;  
Serial.print("ADC = ");  
Serial.print(adc);  
Serial.print(" | Suhu = ");  
Serial.println(suhu);
```

Jika nilai suhu menunjukkan angka yang tidak masuk akal, programmer dapat memeriksa nilai ADC terlebih dahulu. Dengan cara ini, proses debugging dilakukan secara bertahap:



Gambar 24. Tahap Debugging

Pendekatan tersebut lebih efektif dibandingkan langsung menganggap bahwa kesalahan terdapat pada seluruh program.

6.3.4 Debugging pada Aktuator

Aktuator seperti relay, motor DC, servo, dan solenoid juga perlu diuji secara terpisah. Misalnya sebuah relay seharusnya aktif ketika suhu mencapai 35 °C.

Program dapat dibuat sederhana terlebih dahulu:

```
digitalWrite(RELAY, HIGH);
delay(1000);
digitalWrite(RELAY, LOW);
delay(1000);
```

Jika relay tidak bekerja, masalah dapat ditelusuri pada pin mikrokontroler, modul relay, tegangan catu daya, atau konfigurasi *active HIGH/active LOW*.

Setelah perangkat keras dipastikan bekerja, barulah fungsi tersebut digabungkan dengan program sensor dan algoritma kontrol.

6.3.5 Pengukuran Waktu Eksekusi Program

Optimasi kinerja membutuhkan informasi mengenai waktu yang digunakan oleh suatu proses. Arduino menyediakan fungsi `micros()` dan `millis()` untuk membantu pengukuran waktu.

Contoh:

```
unsigned long mulai = micros();

int nilai = analogRead(A0);
float tegangan = nilai * (5.0 / 1023.0);
float suhu = tegangan * 100.0;

unsigned long selesai = micros();

Serial.print("Waktu eksekusi = ");
Serial.print(selesai - mulai);
Serial.println(" us");
```

Dengan cara tersebut, programmer dapat mengetahui berapa lama suatu bagian program dieksekusi.

Pengukuran waktu penting terutama pada sistem yang memiliki persyaratan *real-time*. Jika sebuah proses membutuhkan waktu lebih lama daripada periode yang tersedia, algoritma atau struktur program perlu diperbaiki.

6.3.6 Optimasi Penggunaan Memori

Mikrokontroler memiliki kapasitas memori yang terbatas. Pada Arduino Uno, SRAM hanya sebesar 2 KB. Oleh karena itu, penggunaan memori harus diperhatikan.

Penggunaan array berukuran besar dapat menghabiskan SRAM:

```
char data[500];
```

Jika data tersebut hanya digunakan untuk menyimpan teks konstan, programmer dapat mempertimbangkan mekanisme penyimpanan pada Flash sesuai dukungan platform.

Selain itu, penggunaan tipe data harus disesuaikan dengan kebutuhan.

```
int status = 1;
```

Jika variabel hanya menyimpan nilai 0 atau 1, tipe data yang lebih kecil dapat digunakan:

```
byte status = 1;
```

Penghematan tersebut menjadi semakin penting ketika sistem memiliki banyak variabel.

6.3.7 Optimasi Algoritma

Optimasi tidak selalu berarti mengganti satu instruksi dengan instruksi lain. Perbaikan algoritma sering memberikan pengaruh yang lebih besar.

Sebagai contoh, sebuah sensor tidak perlu dibaca setiap kali loop() berulang jika kebutuhan sistem hanya satu pembacaan setiap 500 ms.

Program yang kurang efisien:

```
void loop() {  
    int sensor = analogRead(A0);  
    prosesSensor(sensor);  
}
```

Program dapat diperbaiki:

```
unsigned long waktuSebelumnya = 0;  
  
void loop() {  
    if (millis() - waktuSebelumnya >= 500) {  
        waktuSebelumnya = millis();  
        int sensor = analogRead(A0);  
        prosesSensor(sensor);  
    }  
}
```

Dengan demikian, mikrokontroler memiliki waktu untuk menjalankan tugas lainnya.

6.3.8 Contoh Penerapan: Optimasi Sistem Monitoring Suhu

Misalkan sebuah sistem menggunakan Arduino Uno, LM35, LCD I2C, dan relay kipas. Sistem harus membaca suhu setiap 500 ms, menampilkan suhu pada LCD setiap 1 detik, dan mengendalikan kipas berdasarkan ambang 35 °C.

Program yang menggunakan banyak `delay()` dapat menyebabkan sistem menjadi tidak responsif.

Contoh struktur yang kurang baik:

```
void loop() {  
    int adc = analogRead(A0);  
    float suhu = adc * 0.488;  
    lcd.print(suhu);  
    delay(1000);  
    if (suhu >= 35) {  
        digitalWrite(RELAY, HIGH);  
    }  
    delay(1000);  
}
```

Masalah program tersebut adalah terdapat waktu tunggu yang menyebabkan mikrokontroler tidak segera memproses tugas lainnya.

Program dapat diperbaiki dengan penjadwalan berbasis `millis()`:

```
unsigned long waktuSensor = 0;  
unsigned long waktuLCD = 0;  
const unsigned long intervalSensor = 500;  
const unsigned long intervalLCD = 1000;  
float suhu = 0;
```

```

void loop() {
    unsigned long sekarang = millis();
    if (sekarang - waktuSensor >= intervalSensor) {
        waktuSensor = sekarang;
        int adc = analogRead(A0);
        float tegangan = adc * (5.0 / 1023.0);
        suhu = tegangan * 100.0;
        if (suhu >= 35.0) {
            digitalWrite(RELAY, HIGH);
        } else {
            digitalWrite(RELAY, LOW);
        }
    }

    // Pembaruan LCD
    if (sekarang - waktuLCD >= intervalLCD) {
        waktuLCD = sekarang;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Suhu: ");
        lcd.print(suhu);
        lcd.print(" C");
    }
}

```

Pada program tersebut, pembacaan sensor dan pembaruan LCD memiliki periode berbeda. Sensor diperbarui setiap 500 ms, sedangkan LCD diperbarui setiap 1 detik. Tidak terdapat `delay()` yang menghentikan keseluruhan program.

Struktur tersebut juga memungkinkan tugas lain ditambahkan dengan mudah, misalnya pengiriman data melalui Wi-Fi, pencatatan data ke kartu SD, atau komunikasi dengan mikrokontroler lain.

6.3.9 Optimasi dengan Histeresis

Pada sistem kontrol, optimasi juga dapat dilakukan pada algoritma untuk mencegah aktuatur bekerja terlalu sering. Jika kipas hanya menggunakan kondisi:

```

if (suhu >= 35) {
    digitalWrite(RELAY, HIGH);
} else {
    digitalWrite(RELAY, LOW);
}

```

ketika suhu berada di sekitar 35 °C, relay dapat mengalami kondisi hidup-mati secara berulang.

Solusinya adalah menggunakan histeresis:

```

if (suhu >= 35) {
    digitalWrite(RELAY, HIGH);
}

if (suhu <= 33) {
    digitalWrite(RELAY, LOW);
}

```

Dengan demikian, kipas menyala pada suhu 35 °C atau lebih dan tetap menyala sampai suhu turun hingga 33 °C. Teknik ini mengurangi frekuensi switching relay dan dapat meningkatkan umur komponen.

DAFTAR PUSTAKA

- Boxall, J. (2013). *Arduino Workshop: A Hands-On Introduction with 65 Projects*. No Starch Press.
- Fraden, J. (2016). *Handbook of Modern Sensors: Physics, Designs, and Applications* (5th ed.). Springer International Publishing.
- Kadir, A. (2018). *Pemrograman Arduino & Processing*. Penerbit Andi.
- Kadir, A. (2019). *Langkah Mudah Belajar Sensor Menggunakan Arduino*. Penerbit Andi.
- Mazidi, M. A., Naimi, S., & Naimi, S. (2014). *The AVR Microcontroller and Embedded Systems: Using Assembly and C* (2nd ed.). Pearson Education. Mazidi, M. A., Naimi, S., & Naimi, S. (2014). *The AVR Microcontroller and Embedded Systems: Using Assembly and C* (2nd ed.). Pearson Education.
- Noerhartati, E., & Santoso, B. (2020). *Dasar Sistem Tertanam dan Antarmuka Mikrokontroler*. Deepublish

TENTANG PENULIS



Ir. Zainal Akbar, M.T. menghabiskan sebagian besar perjalanan hidup serta karirnya di dunia teknik elektro. Minatnya yang mendalam pada teknologi membawanya menempuh studi D-IV Teknik Elektro di Politeknik Negeri Ujung Pandang hingga lulus pada 2014, sebelum akhirnya melanjutkan dan merampungkan studi magister (S-2) Teknik Elektro di Universitas Hasanuddin pada 2020. Baginya, dunia mikrokontroler, elektronika, dan sistem kendali bukan sekadar teori di atas kertas, melainkan ruang bermain untuk merancang berbagai inovasi nyata.

Kini aktif mendedikasikan ilmunya sebagai dosen di Politeknik ATI Makassar di bawah naungan Kementerian Perindustrian. Sehari-hari mengampu mata kuliah seru seputar mikrokontroler, elektronika, komunikasi data, robotika industri, hingga sistem manufaktur diskrit. Tak hanya mengajar di kelas, ia juga memegang sertifikasi sebagai Asesor Kompetensi BNSP, operator sistem robot industri dan aktif mengikuti berbagai pelatihan terkini, mulai dari *Internet of Things* (IoT) hingga *data science*.

Di sela-sela rutinitas akademisnya, beliau dikenal sangat produktif dalam riset terapan dan pengabdian masyarakat. Fokus inovasinya banyak bersinggungan dengan integrasi mikrokontroler dan *Internet of Things*.

Buku bertajuk "Sistem Tersemaat - Arduino, Sensor dan Aplikasi Praktis" ini membahas dasar-dasar pengembangan sistem tersemaat (*embedded systems*) berbasis mikrokontroler Arduino. Pembaca diajak untuk memahami prinsip kerja perangkat keras, mulai dari arsitektur mikrokontroler, penggunaan berbagai jenis sensor, hingga teknik antarmuka komponen elektronik dasar.

Selain penjelasan teoritis, materi dalam tulisan ini difokuskan pada implementasi praktis dan aplikasi nyata di bidang elektronika serta otomatisasi. Melalui berbagai panduan eksperimen dan contoh proyek, dokumen ini dirancang untuk membantu mahasiswa, pelajar, maupun praktisi dalam merancang serta membangun perangkat cerdas berbasis Arduino secara mandiri dan efektif.



Al Arsy Media

Penerbit Buku Umum dan Pendidikan

Alamat : Grand Sulawesi Antang E77

Jl. AMD Antang, Kecamatan Manggala - Makassar

