Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT

KERJA PRAKTIK

Program Studi

S1 Teknik Komputer

Oleh:

FAIRUS FRANS MAULANA PAMBAYUN SUGIARTO

22410200006

FAKULTAS TEKNOLOGI DAN INFORMATIKA

UNIVERSITAS DINAMIKA

2025

Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium

Menggunakan ESP32 dan Buzzer Berbasis MQTT

Diajukan sebagai salah satu syarat untuk menyelesaikan

Program Sarjana

Disusun Oleh:

Nama	**:**	**Fairus Frans Maulana Pambayun Sugiarto**
NIM	**:**	**22410200006**
Program	**:**	**S1(Strata Satu)**
Jurusan	**:**	**Teknik Komputer**

FAKULTAS TEKNOLOGI DAN INFORMATIKA

UNIVERSITAS DINAMIKA

2025

Tetap semangat meskipun dalam keadaan tidak baik- baik Saja

Laporan Kerja Praktik ini

Saya Persembahkan kepada Keluarga Saya Tercinta,

Dosen Pembimbing,

Dan Saya Sendiri

LEMBAR PENGESAHAN

Judul Kerja Praktik

Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium

Menggunakan ESP32 dan Buzzer Berbasis MQTT

Laporan Kerja Praktik

oleh:

Fairus Frans Maulana Pambayun Sugiarto

22410200006

Telah diperiksa, diuji, dan disetujui

Surabaya, 20 November 2025

Disetujui

Dosen Pembimbing	Penyelia,
Dr. Ira Puspasari, S.Si., M.T	**Pauladie Susanto, S.Kom., M.T.**
NIDN. **0710078601**	NIDN. **0729047501**

Mengetahui,

Ketua Program Studi S1 Teknik Komputer

Dr. Ira Puspasari, S.Si., M.T
NIDN. **0710078601**

SURAT PERNYATAAN

PERSETUJUAN PUBLIKASI DAN KEASLIAN KARYA ILMIAH

Sebagai mahasiswa Universitas Dinamika, Saya :

Nama : **Fairus Frans Maulana Pambayun Sugiarto**

NIM : **22410200006**

Program Studi : **S1 Teknik Komputer**

Fakultas : **Teknologi dan Informatika**

Jenis Karya : **Laporan Kerja Praktik**

Judul Karya : **MONITORING NOTIFIKASI JADWAL PEMBERIAN PAKAN IKAN AKUARIUM MENGGUNAKAN ESP32 DAN BUZZER BERBASIS MQTT**

Menyatakan dengan sesungguhnya bahwa :

1. Demi pengembangan ilmu pengetahuan, Teknologi dan Seni, Saya menyetujui memberikan kepada **Universitas Dinamika** Hak Bebas Royalti Non-Eksklusif ***(Non-Exclusive Royalty Free Right)*** atas seluruh isi/sebagian karya ilmiah saya tersebut diatas untuk disimpan, dialihmediakan, dan dikelola dalam bentuk pangkalan data ***(Database)*** untuk selanjutnya didistribusikan atau dipublikasikan demi kepentingan akademis dengan tetap mencantumkan nama saya sebagai penulis atau pencipta dan sebagai pemilik Hak Cipta.
2. Karya tersebut diatas adalah hasil karya asli saya, dan bukan plagiat baik sebagian maupun keseluruhan. Kutipan, karya atau pendapat orang lain yang ada dalam karya ilmiah ini semata-mata hanya sebagai rujukan yang dicantumkan dalam Daftar Pustaka Saya.
3. Apabila dikemudian hari ditemukan dan terbukti terdapat tindakan plagiasi pada karya ilmiah ini, maka saya bersedia untuk menerima pencabutan terhadap gelar kesarjanaan yang telah diberikan kepada saya.

Demikian surat pernyataan ini saya buat dengan sebenar-benarnya.

Surabaya, 11 Desember 2025

Penulis

Fairus Frans Maulana P.S.

NIM. 22410200006

ABSTRAK

Perawatan akuarium adalah salah satu bidang otomatisasi yang telah direvolusi oleh perkembangan *Internet of Things* (IoT). Ketidakteraturan dalam pemberian pakan ikan, baik konsistensi maupun waktu, adalah salah satu masalah umum yang dapat membahayakan kesehatan ikan. Pada Ruang Dosen S1 Teknik Komputer Universitas Dinamika, akuarium sering menjadi bagian dari suasana kerja, sehingga diperlukan sistem yang dapat membantu menjaga pakan teratur. Penelitian kerja praktek ini merancang dan mengimplementasikan sistem yang melacak jadwal pemberian pakan ikan di akuarium menggunakan ESP32 dengan protokol komunikasi MQTT. Sistem ini memiliki konektivitas Wi-Fi yang stabil dan mendukung komunikasi ringan berbasis MQTT, dan dilengkapi dengan buzzer untuk memberi tahu ikan bahwa waktu pemberian pakan telah tiba. Hasil pengujian menunjukkan bahwa sistem mampu memberikan notifikasi tepat waktu dengan rata-rata delay kurang dari 3 detik. Buzzer berhasil memberikan pengingat pada setiap jadwal yang ditetapkan, yang memungkinkan pemberian pakan dilakukan dengan lebih teratur. Oleh karena itu, sistem ini tidak hanya dapat membantu menjaga kesehatan ikan, tetapi juga membuat dosing lebih mudah untuk dilakukan saat memelihara ikan di akuarium.

Kata kunci: *Internet of Things* (IoT), ESP32, MQTT, Buzzer, Monitoring, Notifikasi, Pemberian Pakan Ikan.

KATA PENGANTAR

Dengan mengucapkan puji syukur ke hadirat Tuhan Yang Maha Esa atas segala limpahan rahmat dan hidayah-Nya, penulis dapat menyelesaikan Laporan Kerja Praktik dengan judul "Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT."

Laporan Kerja Praktik ini disusun dalam rangka penulisan laporan untuk lulus mata kuliah Kerja Praktik pada Program Studi S1 Teknik Komputer Universitas Dinamika Surabaya. Dalam penulisan Laporan Kerja Praktik ini tidak lepas dari adanya bimbingan, nasihat, bantuan, saran, serta motivasi yang diberikan kepada oleh pihak terkait. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Orang tua saya yang selalu memberikan Doa, Dukungan dan Motivasi selama mengikui kegiatan .
2. Bapak Pauladie Susanto, S.Kom., M.T., Selaku penyelia sekaligus Ketua Program Studi S1 Teknik Komputer lama yang telah memberikan izin kepada penulis untuk melakukan Kerja Praktik di Ruang Dosen S1 Teknik Komputer .
3. Ibu Dr. Ira Puspasari, S.Si., M.T., selaku Ketua Program Studi S1 Teknik Komputer baru sekaligus dosen pembimbing yang telah memberikan arahan dan bimbingan selama proses kerja praktik dan peyusunan laporan ini.
4. Ibu Elisabeth Ria Anggreani A.Md.Keb., selaku koordinator kerja praktik di Universitas Dinamika.
5. Bapak Charisma Dimas Affandi, S.T., selaku laboran yang telah memberikan bimbingan dan bantuan dalam mengerjakan Project Akhir Kerja Praktik di Universitas Dinamika.

Pihak-pihak lain yang tidak dapat disebutkan satu-persatu yang telah memberikan bantuan dan dukungan kepada penulis.

Semoga Allah SWT memberikan balasan yang setimpal kepada semua pihak yang telah membantu dan memberikan bimbingan serta nasehat dalam proses Kerja Praktik ini. Penulis menyadari bahwa Kerja Praktik yang dikerjakan ini masih banyak terdapat kekurangan sehingga kritik yang bersifat membangun dan saran

dari semua pihak sangatlah diharapkan agar aplikasi ini dapat diperbaiki menjadi lebih baik lagi. Semoga laporan ini dapat memberikan manfaat dan menjadi referensi bagi pihak-pihak yang membutuhkan

Surabaya, 20 November 2025

Fairus Frans Maulana Pambayun Sugiarto

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi *Internet of Things* (IoT) telah mempermudah banyak aspek kehidupan, seperti menjaga ikan hias di akuarium. Pakan yang tidak teratur, baik dalam jumlah maupun waktu, adalah salah satu masalah umum yang sering dihadapi, yang dapat berdampak negatif pada kesehatan ikan dan lingkungan akuarium. Seringkali, di lingkungan akademik, terutama di Ruang Dosen S1 Teknik Komputer Universitas Dinamika, ada akuarium untuk kenyamanan dan hiasan. Namun, karena banyaknya aktivitas dosen, pemberian pakan terkadang terlupakan atau tidak teratur. Akibatnya, diperlukan sebuah sistem yang dapat melakukan monitoring dan sekaligus memberikan notifikasi secara otomatis tentang jadwal pemberian pakan ikan.

Karena memiliki konektivitas Wi-Fi, daya komputasi yang tinggi, dan kompatibel dengan protokol komunikasi ringan MQTT, ESP32 adalah mikrokontroler yang ideal untuk digunakan. Dengan menambahkan buzzer, sistem dapat memberikan notifikasi suara ketika jadwal pemberian pakan tiba. Ini memudahkan dosen untuk memastikan ikan mendapatkan pakan tepat waktu. Studi sebelumnya menunjukkan bahwa penggunaan *Internet of Things* (IoT) dalam sistem pemberian pakan ikan otomatis dapat berhasil. Chaidir dkk. (2024) mengontrol alat pemberi pakan dengan delay komunikasi yang rendah dengan menggunakan ESP32 dan MQTT.

Sementara itu, Burhani dkk. (2022) menerapkan sistem IoT dan sensor untuk memantau kualitas air dan pakan otomatis. Penelitian lain oleh Koromari dan David (2023) merancang sistem pakan otomatis sekaligus monitoring TDS berbasis ESP32 dan MQTT, yang menunjukkan keberhasilan implementasi dalam hal pemeliharaan ikan hias di akuarium. Hasil penelitian ini menunjukkan bahwa integrasi buzzer, ESP32, dan MQTT adalah solusi yang layak untuk dikembangkan sebagai solusi untuk melacak jadwal pemberian pakan ikan di ruang dosen.

1.2 Rumusan Masalah

Berdasarkan informasi di atas, rumusan masalah yang dapat diidentifikasi adalah sebagai berikut:

1. Bagaimana merancang sistem pemantauan otomatis untuk pemberian pakan ikan yang dapat diakses melalui jaringan Wi-Fi?
2. Apa saja bagian yang diperlukan untuk menerapkan sistem ini?
3. Bagaimana cara kerja bagian-bagian dalam sistem yang mengawasi pencarian pakan ikan?
4. Dengan cara apa sistem akan memberi tahu pengguna tentang jadwal pemberian pakan?
5. Seberapa efektif sistem ini dalam meningkatkan jumlah pakan yang diberikan kepada ikan di akuarium ruang dosen secara teratur?

1.3 Batasan Masalah

Batasan masalah yang ditetapkan adalah sebagai berikut:

1. Ruang Lingkup Sistem: Sistem yang dikembangkan hanya akan diterapkan pada akuarium ikan hias di Ruang Dosen S1 Teknik Komputer Universitas Dinamika, untuk memastikan fokus dan kelayakan penelitian. Jenis akuarium lain tidak akan diuji atau diimplementasikan.

2. Komponen Sistem: Penelitian ini tidak akan mengintegrasikan komponen sistem lainnya, seperti sensor kualitas air, tetapi akan menggunakan mikrokontroler ESP32, buzzer, dan modul komunikasi MQTT.

3. Notifikasi: Notifikasi hanya akan dikirim melalui buzzer suara. Tidak akan ada notifikasi visual atau penggunaan aplikasi *mobile* lainnya.

4. Jadwal Pemberian Pakan: Pengguna akan menentukan jadwal pemberian pakan secara manual, dan tidak akan menyertakan algoritma pembelajaran untuk penyesuaian otomatis berdasarkan pola makan ikan.

5. Efektivitas dan Kualitas: Penelitian ini akan membahas efektivitas sistem dalam hal konsistensi pemberian pakan, tetapi tidak akan membahas dampak jangka panjang terhadap kesehatan ikan dan kualitas air, yang mungkin akan menjadi topik penelitian lain di masa mendatang.

1.4 Tujuan

Tujuan penelitian ini adalah sebagai berikut:

1. Merancang Sistem Monitoring: Menciptakan sistem monitoring otomatis untuk pemberian pakan ikan yang dapat diakses melalui jaringan Wi-Fi, memudahkan pengguna untuk mengontrol jadwal pemberian pakan.
2. Identifikasi Komponen: Mengidentifikasi dan mendokumentasikan komponen yang diperlukan untuk implementasi sistem, serta menjelaskan cara masing-masing komponen bekerja.
3. Pengembangan Notifikasi: Buat sistem yang efektif dengan buzzer untuk mengingatkan pengguna saat jadwal pemberian pakan tiba.
4. Evaluasi Efektivitas: Mengevaluasi seberapa efektif sistem ini dapat meningkatkan konsistensi pemberian pakan ikan di akuarium ruang dosen dan mengevaluasi tanggapan pengguna tentang penggunaan sistem.
5. Dampak terhadap Kesehatan Ikan: Melakukan analisis awal tentang efek penggunaan sistem ini terhadap kesehatan ikan dan kualitas lingkungan akuarium, meskipun hal ini tidak dilakukan secara menyeluruh dalam penelitian ini.

1.5 Manfaat

Manfaat yang diharapkan dari penelitian ini adalah sebagai berikut:

1. Kemudahan dalam Pemeliharaan: Sistem ini akan membantu pemilik akuarium memberi pakan ikan secara teratur, sehingga mengurangi risiko kesehatan ikan karena pakan yang tidak teratur.
2. Monitoring Kualitas Air: Adanya sistem monitoring akan membuat pengguna lebih mudah melacak kondisi lingkungan akuarium, termasuk kualitas air, yang merupakan komponen penting dalam menjaga ikan.
3. Efisiensi Waktu: Pemilik ikan dapat menghemat waktu dengan sistem otomatis, terutama bagi mereka yang sibuk, karena mereka tidak perlu memikirkan pakan yang terlupakan.
4. Peningkatan Pengetahuan: Penelitian ini diharapkan dapat memberikan wawasan baru tentang penerapan teknologi *Internet of Things* (IoT) dalam akuakultur. Hasil penelitian ini akan menjadi dasar untuk penelitian selanjutnya.

5. Kontribusi pada Sektor Perikanan: Diharapkan dapat berdampak positif pada ekonomi, terutama bagi komunitas yang terlibat dalam budidaya ikan hias, dengan meningkatkan efisiensi dalam pemberian pakan dan pemeliharaan ikan.
6. Pengembangan Teknologi: Penelitian ini akan mendorong kemajuan teknologi di bidang otomasi dan kontrol, serta aplikasi IoT dalam kehidupan sehari-hari.

BAB II

GAMBARAN UMUM PERUSAHAAN

2.1 Sejarah Universitas Dinamika

A. 30 April 1983, Pengembangan teknologi dan informasi menjadi hal penting dalam pembangunan dan pengembangan nasional. Kedua hal tersebut juga harus diiringi dengan di bidang ekonomi dan bisnis untuk bisa bersaing di era yang terus berkembang. Seni dan budaya harus tetap di pertahankan agar identitas bangsa tidak musnah. Melalui empat (4) hal utama, yaitu kritis, kreatif, kolaborasi, dan komunikasi, para pendiri yang terdiri dari laksda. TNI (Purn) Mardiono, Ir. Andrian A.T, Ir. Handoko A.T, Dra. Rosy Merianti, Ak. dalam bidang teknologi informasi dengan nama AKIS (Akademi Komputer dan Informatika Surabaya).
B. 10 Maret 1984, Izin operasional penyelenggara program Diploma III Manajemen Informatika diberikan kepada AKIS melalui SK Kopertis Wilayah VII Jawa Timur.
C. 19 Juni 1984, AKIS yang berlokasi di Ketintang Surabaya memperoleh status terdaftar dari DIKTI.
D. 20 Maret 1986, Terus meningkatnya kebutuhan pendidikan, Yayasan Putra Bhakti memutuskan untuk merubah Akademi menjadi Sekolah Tinggi. AKIS (Akademi Komputer dan Informatika Surabaya) berubah menjadi Sekolah Tinggi Manajemen Informatika dan Teknik Komputer Surabaya, yang lebih dikenal dengan STIKOM Surabaya.
E. 11 Desember 1987, STIKOM Surabaya membangun kampus pertama yang berlokasi di jalan Kutisari No.66 Surabaya, yang diresmikan oleh Letnan Jendral TNI Wahono selaku Gubernur Jawa Timur pada saat itu.
F. 28 Oktober 1997, Awal pemasangan tiang pancang pertama STIKOM Surabaya di Jalan Raya Kedung Baruk No.98 Surabaya bersamaan dengan Hari Sumpah Pemuda.
G. 04 September 2014, Seiring dengan perubahan zaman serta kebutuhan masyarakat, STIKOM Surabaya resmi berubah menjadi Institut dengan

nama Institut Bisnis dan informatika STIKOM Surabaya yang memiliki 2 fakultas dengan 9 program studi.

H. 29 Juli 2019, Melalui Surat Keputusan Riset Dikti, Institut Bisnis dan Informatika STIKOM Surabaya resmi berubah menjadi Universitas Dinamika yang memiliki 2 fakultas dengan 9 program studi, yakni Fakultas, Prodi S1 Teknik Komputer, Prodi S1 Desain Komunikasi Visual, Prodi S1 Desain Produk, Prodi D4 Produksi Film dan Televisi, dan Prodi D3 Sistem Informasi. Serta Fakultas Ekonomi dan Bisnis (FEB) dengan Prodi S1 Manajemen, Prodi S1 Akuntansi, dan Prodi D3 Administrasi Perkantoran.

I. 31 Mei 2021, Melalui Surat Keputusan Rektor, Universitas Dinamika melakukan perubahan struktur organisasi dengan membentuk fakultas baru, yakni Fakultas Desain dan Industri Kreatif (FDIK) dengan 3 program studi, yaitu Prodi S1 Desain Produk, Prodi S1 Desain Komunikasi Visual, dan D4 Produksi Film dan Televisi yang sebelumnya berada dibawah naungan Fakultas Teknologi dan Informatika (FTI).

2.2 Visi Misi Dan Tujuan Perusahaan

2.2.1 Visi

Menjadi *smart entrepreneurial university* berskala global yang produktif dalam berinovasi.

2.2.2 Misi

1. Menyelenggarakan dan mengembangkan pendidikan berbasis teknologi informasi yang bermutu dan berdaya saing global.
2. Melaksanakan penelitian yang berfokus pada pengembangan inovasi untuk mewujudkan entrepreneurial university.
3. Melakukan pengabdian untuk menyebarluaskan ipteks dan hasil inovasi bagi kesejahteraan masyarakat.
4. Melaksanakan kemitraan berskala global.
5. Mengembangkan bisnis dan kewirausahaan secara otonom yang akuntabel dan transparan.

2.2.3 Tujuan

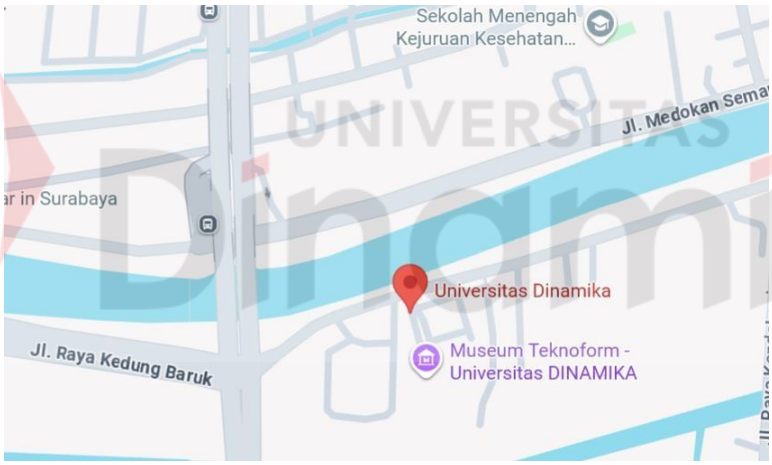
1. Menyelenggarakan pendidikan yang berkualitas, inovatif, dan futuristik.

2. Menciptakan SDM berdaya saing global dan berjiwa entrepreneur.
3. Menghasilkan penelitian berkualitas dan berskala global.
4. Menghasilkan inovasi yang bernilai jual dan bermanfaat bagi masyarakat.
5. Melaksanakan diseminasi ipteks dan/atau hasil inovasi untuk meningkatkan kesejahteraan masyarakat.
6. Mewujudkan kemitraan berskala global.
7. Menjamin keberlanjutan Perguruan Tinggi.

2.3 Profil Perusahaan

Nama Instansi : Ruang Prodi S1 Teknik Komputer Universitas Dinamika

Alamat : Jl. Raya Kedung Baruk No. 98, Kedung Baruk, Kec. Rungkut, Surabaya, Jawa Timur 60298



Gambar 2.3 Lokasi Ruang Prodi S1 Teknik Komputer Universitas Dinamika (Sumber: https://www.google.com/maps/)

Email : universitasdinamika@dinamika.ac.id

Website : www.dinamika.ac.id

No Telfon & Faks: (031)8721731 / 8710218

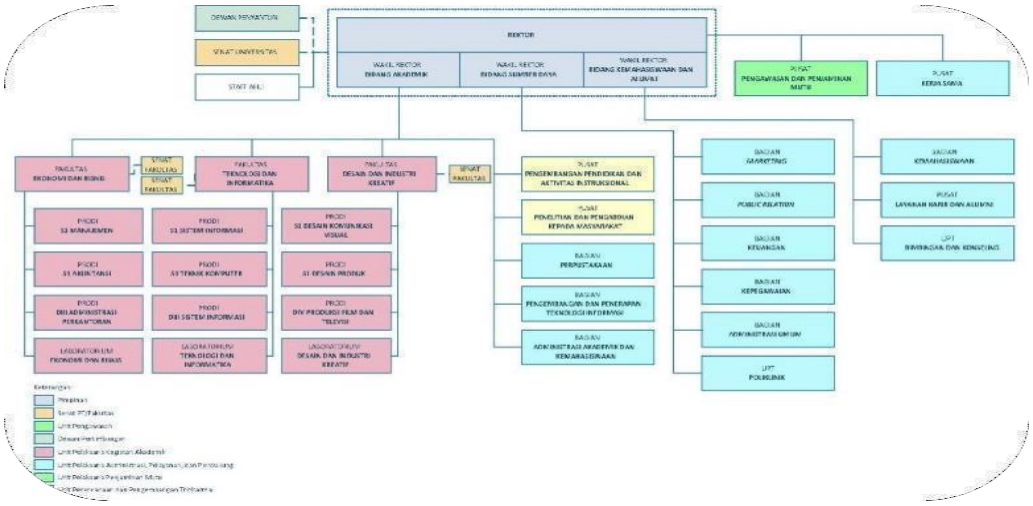
Sosial Media

Facebook : Universitas Dinamika

Youtube : Universitas Dinamika

Instagram : @universitasdinamika

2.4 Struktur Organisasi



Gambar 2.4 Struktur Organisasi

Lingkaran merah pada gambar tersebut menunjukkan Ruang Prodi S1 Teknik Komputer, yang berada di bawah Fakultas Teknologi dan Informatika, Universitas Dinamika. Di Ruang Prodi inilah tempat melaksanakan kerja praktik untuk merancang Simulasi Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP 32 dan Buzzer Sebagai Alat perangkat Hardwarenya. Untuk perangkat software nya Memantau kapan harus memberi pakan ikan Akuarium Menggunakan MQTT sebagai pengontrol Jadwal Pemberian pakan Ikan Akuarium.

2.5 Program Studi S1 Teknik Komputer

Program Studi S1 Teknik Komputer mulai beroperasi sejak tahun 1991. Tujuan dari program ini adalah menghasilkan lulusan yang memiliki kompetensi di bidang sistem komputer dan otomasi industri. Mahasiswa dibekali dengan berbagai mata kuliah pendukung, seperti Mikrokontroler, *Programmable Logic Controller* (PLC), Sistem Digital, Jaringan Komputer, serta Robotika. Lulusan dari program ini diharapkan mampu melakukan analisis, perancangan, dan pengembangan sistem otomasi industri yang berbasis komputer. Karier yang dapat dijalani antara lain sebagai *Computer System Engineer, Embedded System Developer,* dan *Automation Engineer.*

Program ini juga memberikan penekanan pada keseimbangan antara pemahaman teoritis dan keterampilan praktis melalui kegiatan seperti praktikum,

penelitian, dan proyek akhir. Mahasiswa juga dibekali kemampuan dalam pemrograman tingkat lanjut, pengolahan data, serta pengembangan sistem berbasis kecerdasan buatan (AI). Kurikulum yang disusun selalu diperbarui agar selaras dengan perkembangan kebutuhan industri, sehingga lulusan S1 Teknik Komputer Universitas Dinamika mampu bersaing di dunia kerja maupun melanjutkan pendidikan ke jenjang lebih tinggi.

Selain itu, mahasiswa diberikan peluang untuk mengikuti sertifikasi industri seperti *Cisco Certified Network Associate* (CCNA) dan sertifikasi di bidang embedded system programming, guna meningkatkan daya saing di dunia profesional. Melalui kegiatan kerja praktik, mahasiswa dapat mengembangkan keterampilan sesuai kebutuhan industri dan memperoleh pengalaman nyata sebelum memasuki dunia kerja secara langsung.

2.5.1 VISI MISI

Visi:

Mengembangkan keilmuan di bidang IoT yang didukung oleh kecerdasan artifisial, dan diintegrasikan dengan konsep *technopreneurship*, sehingga mampu menciptakan inovasi yang bermanfaat bagi masyarakat dan industri berskala global.

Misi:

1. Mengembangkan pendidikan dan pengajaran di bidang Teknik Komputer yang bermutu, berwawasan global, dan mengarah pada *technopreneurship*.
2. Melaksanakan penelitian di bidang Teknik Komputer yang inovatif dan solutif bagi masyarakat dan industri berskala global.
3. Melaksanakan pengabdian atau penerapan hasil inovasi di bidang Teknik Komputer yang bermanfaat bagi masyarakat dan industri.

2.5.2 TUJUAN

Tujuan Program Studi S1 Teknik Komputer adalah sebagai berikut:

1. Lulusan memiliki kemampuan menganalisis permasalahan sistem komputer khususnya pada aspek perangkat lunak dan perangkat keras untuk menghasilkan solusi bagi organisasi.

2. Lulusan memiliki kemampuan menganalisis perangkat lunak (meliputi pemrograman antarmuka, pemrograman *real-time*) dan perangkat keras (meliputi pemantauan, pengendalian) sistem komputer sebagai solusi bagi permasalahan organisasi.
3. Lulusan memiliki kemampuan menganalisis dan merancang sistem komputer dengan menerapkan sistem tertanam, *Internet of Things* (IoT), kecerdasan artifisial, dan/atau jaringan komputer untuk menghasilkan solusi bagi organisasi.
4. Lulusan yang memiliki kemampuan dalam merumuskan keputusan yang tepat berdasarkan analisis informasi dan data, beretika, dan bertanggung jawab pada pekerjaan dalam lingkup tugasnya.

2.5.3 PROFESI LULUSAN

Profesi Lulusan Program Studi

1. IoT *Engineer* **:** Menyediakan produk dan atau solusi IoT sesuai dengan kebutuhan penggguna.
2. *Artificial Intelligent Engineer* **:** Membangun solusi berbasis kecerdasaan artifisial (*Artificial Intelligence*/AI)
3. *Network Designer* **:** Melaksanakan penyediaan desain instalasi jaringan dan infrastruktur meliputi kegiatan pemetaan kebutuhan, monitoring dan pengawasan dampak design pembangunan dan pengembangan instalasi jaringan dan infrastruktur yang dibutuhkan oleh user sejalan dengan rencana dan pengembangan organisasi.
4. *Digital Computer Technology Advisor* **:** Memecahkan masalah teknis, memberikan saran tentang perangkat keras dan perangkat lunak yang tepat, serta mengoptimalkan penggunaan teknologi dalam bisnis atau kehidupan sehari-hari.
5. *Industrial Automation* **:** Meningkatkan efisiensi dan produktivitas sistem produksi di industri

BAB III

LANDASAN TEORI

3.1 Akuarium Hias

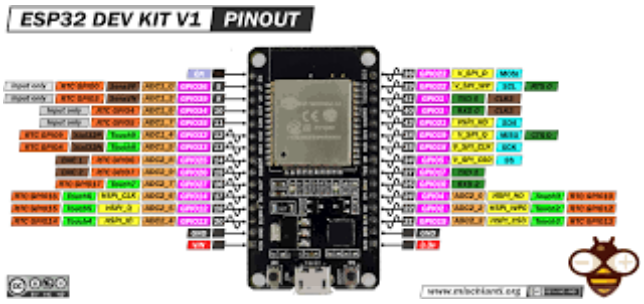
Gambar 3.1 Akuarium Hias

Akuarium hias adalah tempat buatan di mana ikan, tanaman air, dan dekorasi lainnya dipelihara untuk rekreasi, pendidikan, dan penelitian. Menurut Wijaya dan Wellem (2022), akuarium kontemporer tidak hanya berfungsi sebagai sarana estetika, tetapi juga telah berkembang menjadi alat untuk mengajar dan mengeksplorasi teknologi, khususnya *Internet of Things* (IoT). Akuarium hias mampu memperkenalkan konsep ekosistem perairan dalam ruang terbatas, yang membuatnya berguna sebagai sarana edukasi. Dari perspektif teknologi, akuarium juga menjadi tempat untuk mencoba berbagai inovasi. Misalnya, mereka dapat memantau kualitas air, suhu, kadar oksigen, dan merancang sistem pemberian pakan otomatis. Dengan kemajuan teknologi *Internet of Things* (IoT), akuarium sekarang lebih dari sekadar tempat untuk menyimpan ikan. Mereka sekarang dapat dihubungkan ke sensor, aktuator, dan sistem kendali berbasis mikrokontroler seperti ESP32. Dengan integrasi ini, jaringan internet memungkinkan pemantauan dan pengendalian parameter secara *real-time*.

Dengan membangun akuarium pintar, mahasiswa belajar tentang penggunaan IoT untuk pengawasan dan otomasi. Penggunaan sensor untuk memantau waktu pemberian pakan, pengaturan pencahayaan, dan pengiriman

notifikasi melalui platform MQTT (*Message Queuing Telemetry Transport*) adalah beberapa contohnya. Oleh karena itu, akuarium yang indah tidak hanya membantu dalam hal estetika dan hiburan, tetapi juga membantu dalam pengembangan teknologi cerdas yang berkaitan dengan pendidikan, penelitian, dan penerapan masyarakat.

3.2 ESP 32



Gambar 3.2 ESP 32 DEVKIT V1 beserta *PINOUT*
Sumber: (https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/)

ESP32 merupakan mikrokontroler generasi lanjut yang dikembangkan oleh Espressif Systems sebagai penerus dari ESP8266 dengan kemampuan yang lebih unggul. ESP32 mengintegrasikan prosesor *dual-core* Tensilica LX6 berkecepatan hingga 240 MHz, memori SRAM, serta mendukung konektivitas Wi-Fi 802.11 b/g/n dan Bluetooth v4.2 (*Classic* dan BLE). Fitur-fitur ini menjadikan ESP32 sangat sesuai digunakan dalam proyek *Internet of Things* (IoT), termasuk pada sistem monitoring dan kontrol perangkat cerdas berbasis MQTT.

Menurut Mischianti (n.d.), DOIT ESP32 Dev Kit V1 adalah salah satu varian papan pengembangan yang banyak digunakan dalam penelitian IoT karena menyediakan jumlah *General Purpose Input Output* (GPIO) yang melimpah, mendukung antarmuka komunikasi digital (UART, SPI, I2C), serta mampu beroperasi dengan konsumsi daya rendah (*low power consumption*). Selain itu, penelitian ini juga mempelajari datasheet resmi ESP32 sebagai acuan teknis,

sehingga implementasi sistem sesuai dengan kemampuan dan keterbatasan perangkat.

Dalam konteks sistem pemberian pakan ikan otomatis, ESP32 dipilih karena:

1. Kemampuan Wi-Fi terintegrasi, sehingga dapat langsung terhubung ke broker MQTT tanpa memerlukan modul tambahan (Chaidir dkk. 2024).
2. Dukungan multitasking, yang memungkinkan ESP32 menjalankan proses monitoring sensor dan mengontrol aktuator (seperti buzzer) secara bersamaan.
3. Fleksibilitas antarmuka, sehingga dapat diintegrasikan dengan sensor kualitas air, sensor pakan, hingga aktuator servo untuk sistem pemberian pakan (Burhani dkk. 2022).
4. Efisiensi energi, yang penting untuk sistem monitoring jangka panjang pada perangkat IoT (Nurhidayah dkk. 2024).

Beberapa penelitian terdahulu membuktikan peran penting ESP32 dalam pengembangan sistem *smart aquarium*. Ma'shumah dkk. (2024) menggunakan ESP32 terintegrasi dengan aplikasi Blynk untuk monitoring pakan ikan hias. Wijaya dan Wellem (2022) juga memanfaatkan ESP32 dalam implementasi *Smart Aquarium* yang mengoptimalkan pemantauan kondisi akuarium secara *real-time*. Sementara Koromari dan David (2023) merancang sistem pakan otomatis sekaligus monitoring TDS berbasis ESP32 dan MQTT, yang menunjukkan kinerja baik dalam menjaga keteraturan pemberian pakan ikan hias.

3.3 Buzzer *Low Level Trigger*

Gambar 3.3 Buzzer *Low Level Trigger*
Sumber: (https://www.amazon.in/DAOKAI-Active-Buzzer-Trigger-Arduino/dp/B0B5D6NDM2)

Buzzer adalah komponen elektronika yang dapat menghasilkan suara ketika diberikan tegangan listrik. Buzzer terbagi menjadi dua jenis utama: buzzer aktif (dapat berbunyi hanya ketika diberi tegangan) dan buzzer pasif (membutuhkan sinyal frekuensi untuk menghasilkan suara). Ketika pin *input* (I/O) diberi logika rendah (*LOW* = 0), rangkaian internal akan menjadi lemah, sehingga buzzer akan aktif (menyala atau berbunyi). Dengan kata lain, jika pin kontrol ESP32 diberi sinyal *LOW*, buzzer akan berbunyi dan akan mati.

3.4 *Internet of Things* (IoT)

Paradigma teknologi yang dikenal sebagai *Internet of Things* (IoT) memungkinkan perangkat elektronik terhubung satu sama lain melalui jaringan internet, memungkinkan pertukaran data yang otomatis tanpa intervensi manusia secara langsung. Perangkat fisik IoT dapat dihubungkan dengan sensor, aktuator, dan sistem komunikasi, sehingga proses monitoring dan kontrol menjadi lebih mudah (Koromari & David, 2023). IoT telah banyak dikembangkan dalam budidaya ikan, terutama untuk sistem pakan otomatis dan pemantauan lingkungan perairan. Nurhidayah dkk. (2024) mengembangkan sistem yang menggunakan *Internet of Things* (IoT) untuk memantau kualitas air dan memberikan pakan kepada ikan lele. Penelitian mereka menunjukkan bahwa penggunaan *Internet of Things* (IoT) dapat membantu mengatur jadwal pemberian pakan dan kualitas air yang stabil.

Selain itu, Ma'shumah dkk. (2024) mengembangkan sistem yang mengawasi pemberian pakan ikan hias dengan aplikasi Blynk yang terintegrasi dengan *Internet of Things* (IoT). Sistem ini memungkinkan pengguna mengontrol jadwal pemberian pakan secara *real-time* melalui aplikasi *mobile*, meningkatkan efisiensi dan fleksibilitas dalam pengelolaan akuarium. Selain itu, Wijaya dan Wellem (2022) telah menerapkan IoT pada akuarium pintar, yang memungkinkan sistem untuk secara otomatis memberikan pakan ikan dan mengontrol kondisi lingkungan akuarium. Hal ini menunjukkan bahwa perangkat mikrokontroler dan *Internet of Things* (IoT) dapat menghasilkan solusi kreatif yang bermanfaat dalam bidang akuakultur kontemporer.

3.5 IOT MQTT PANEL

Gambar 3.5 IOT MQTT PANEL
Sumber:
(https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod&hl=id)

Dengan menggunakan protokol *Message Queuing Telemetry Transport* (MQTT), aplikasi mobile berbasis Android IoT MQTT Panel membantu komunikasi antara perangkat *Internet of Things* (IoT) dan pengguna. Aplikasi ini berfungsi sebagai *client*, memungkinkan pengguna untuk mengirim perintah kontrol dan menerima *subscribe* untuk data monitoring dari perangkat IoT. Dengan IoT MQTT Panel, pengguna dapat melihat data sensor, melacak status perangkat, dan mendapatkan notifikasi secara *real-time.*

Dalam penelitian yang dilakukan oleh Nurhidayah dkk. (2024), sistem otomatis yang mengawasi kualitas air dan jadwal pemberian pakan ikan lele terbukti mampu menjaga stabilitas kualitas lingkungan dan mengatur jadwal pemberian pakan untuk budidaya ikan lele, menunjukkan betapa pentingnya IoT diintegrasikan dengan aplikasi pendukung untuk memantau dan mengontrol. Studi Ma'shumah dkk. (2024) menggunakan aplikasi Blynk untuk sistem pemberian pakan ikan hias berbasis IoT, yang memungkinkan pengguna menggunakan perangkat mobile mereka untuk mengatur jadwal pakan. Panel MQTT IoT menawarkan kontrol jarak jauh dan kemudahan integrasi dengan protokol MQTT.

Wijaya dan Wellem (2022) membuat ide untuk akuarium pintar yang berbasis IoT yang secara otomatis mengatur pemberian pakan dan memantau kondisi lingkungannya. Menurut penelitian ini, aplikasi berbasis *Internet of Things* (IoT) dapat menjadi cara kreatif untuk meningkatkan efisiensi pengelolaan akuarium. Dengan menggunakan mikrokontroler, Koromari dan David (2023) membuat sistem pakan otomatis dan pengawasan TDS untuk akuarium ikan hias berbasis IoT. Hasil penelitian menunjukkan bahwa mikrokontroler dapat meningkatkan efisiensi dan reliabilitas pengelolaan akuarium dan kolam ikan dengan menggabungkannya dengan sistem *Internet of Things* (IoT).

3.6 Arduino IDE

Gambar 3.6 Arduino IDE
Sumber: (https://www.arduino.cc/en/software/)

Perangkat lunak *open-source* yang disebut Arduino IDE memungkinkan Anda menulis, mengompilasi, dan mengunggah program ke papan mikrokontroler, seperti Arduino dan ESP32. Arduino IDE juga memiliki pustaka yang mendukung berbagai perangkat keras, dan *compiler*, yang membantu Anda mengembangkan sistem berbasis mikrokontroler.

Dalam penelitian ini, Arduino IDE digunakan untuk menulis program pengendalian buzzer dan melakukan komunikasi data melalui protokol MQTT pada modul ESP32. Ini memenuhi kebutuhan untuk sistem pemantauan jadwal pemberian pakan ikan, di mana ESP32 berfungsi sebagai pusat pengendali yang menetapkan waktu bunyi buzzer sebagai notifikasi.

Beberapa keunggulan Arduino IDE adalah sebagai berikut:

1. Mudah digunakan bahkan oleh pemula, dengan sintaks sederhana berbasis bahasa C/C++;
2. Kompatibilitas luas, mendukung berbagai board mikrokontroler, seperti Arduino Uno, Mega, ESP8266, dan ESP32.
3. *Library* lengkap yang memudahkan integrasi dengan sensor, aktuator, dan protokol komunikasi seperti WiFi dan MQTT.
4. Dapat digunakan di berbagai platform, termasuk *Windows*, Linux, dan *macOS*.

Arduino IDE dapat digunakan untuk menerapkan sistem Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium untuk:

A. Menulis kode program untuk mengatur interval bunyi buzzer dan durasi bunyi sesuai kebutuhan.
B. Konfigurasi koneksi WiFi dan MQTT untuk memungkinkan ESP32 mengirim dan menerima data dari server broker.
C. Upload program ke ESP32 untuk memungkinkan sistem bekerja secara otomatis dan fleksibel.

Studi sebelumnya menggunakan Arduino IDE dan mikrokontroler untuk sistem pemberian pakan ikan otomatis, yang juga menggunakan teknologi *Internet of Things* (IoT) (Ma'shumah, Pramarthaningthyas, & Rohman, 2024). Penggunaan Arduino IDE dalam penelitian ini menunjukkan bahwa Arduino IDE tidak hanya mendukung pembuatan sistem monitoring dasar, tetapi juga dapat diintegrasikan dengan teknologi IoT untuk aplikasi di dunia nyata.

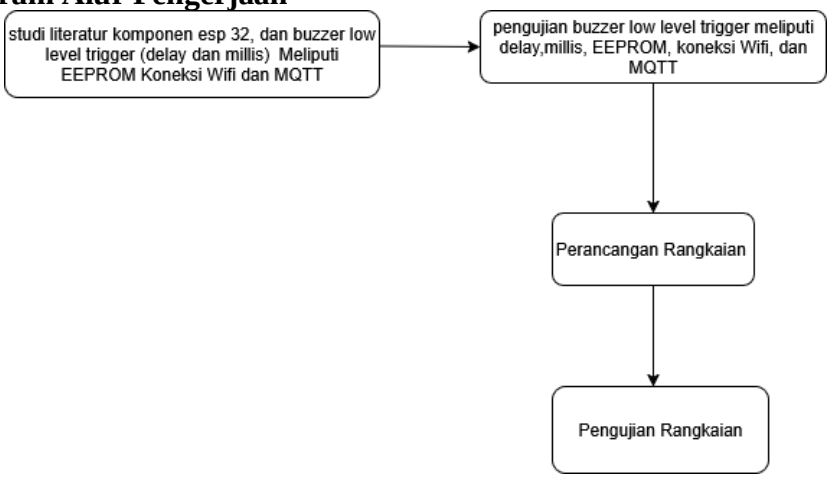
BAB IV

DESKRIPSI PEKERJAAN

4.1. Uraian Pekerjaan

Dalam proyek "Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT", tugas-tugas yang harus diselesaikan dijelaskan dalam uraian pekerjaan ini. Dibagi menjadi beberapa tahapan utama untuk melaksanakan kegiatan kerja praktik. Tahap pertama adalah membaca literatur. Ini mencakup mempelajari bagian dan ide yang digunakan, seperti EEPROM, koneksi Wi-Fi dengan ESP32, dan protokol komunikasi MQTT. Untuk memastikan bahwa komponen berfungsi sesuai dengan perancangan, pengujian lanjutan dilakukan terhadap *buzzer low level trigger*. Setelah tahap pengujian komponen selesai, proses dilanjutkan ke tahap perancangan proyek akhir. Tahap ini mencakup pemrograman mikrokontroler ESP32, proses perakitan perangkat keras, dan integrasi dengan sistem komunikasi MQTT. Untuk memastikan bahwa proyek dapat beroperasi dengan baik sesuai dengan tujuan yang telah ditetapkan, yaitu mengirimkan notifikasi jadwal pemberian pakan ikan secara otomatis melalui koneksi ke sistem, tahap berikutnya adalah pengujian sistem secara keseluruhan.

4.2 Diagram Alur Pengerjaan



Gambar 4.2 Diagram Alur pengerjaan

Diagram alur pekerjaan dari tugas praktik berjudul "Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT di Ruang Dosen S1 Teknik Komputer Universitas Dinamika" terdiri dari beberapa tahapan, yaitu:

4.2.1 Studi Literatur Komponen

Pada tahap awal, informasi tentang fitur-fitur utama yang digunakan dalam sistem dikumpulkan. Fitur-fitur tersebut meliputi:

a) ESP32 → sebuah mikrokontroler yang terintegrasi dengan WiFi dan Bluetooth yang mendukung aplikasi *Internet of Things* (IoT) serta komunikasi berbasis protokol MQTT

b) Buzzer *Low Level Trigger* → buzzer yang aktif saat menerima logika rendah, sehingga dapat dikendalikan langsung oleh pin digital ESP32.

c) MQTT (*Message Queuing Telemetry Transport*) → protokol komunikasi ringan dengan mekanisme *publish/subscribe* yang sangat efisien untuk sistem *Internet of Things* (IoT).

Tahap literatur ini sangat penting untuk memastikan bahwa perangkat lunak dan perangkat keras dapat berinteraksi sesuai kebutuhan sistem.

4.2.2 Pengujian Buzzer *Low Level Trigger*

Sebelum perancangan sistem dilakukan, buzzer diuji secara terpisah dengan metode berikut:

a) `delay()` → Untuk menguji fungsi nyala-mati buzzer secara sederhana (blocking).
b) `millis()` → Untuk pengaturan waktu non-blocking, sehingga ESP32 tetap dapat menjalankan tugas lain.
c) EEPROM → Digunakan untuk menyimpan konfigurasi jadwal pemberian pakan agar data tetap tersimpan walaupun ESP32 dimatikan.
d) Koneksi WiFi → ESP32 dihubungkan ke jaringan WiFi kampus Universitas Dinamika.
e) Koneksi MQTT → ESP32 diuji untuk melakukan *publish* dan *subscribe* pesan, misalnya pada topik "esp32/buzzer" untuk memicu buzzer.

Hasil pengujian menunjukkan buzzer dapat berbunyi sesuai perintah dari broker MQTT, sehingga komunikasi antara perangkat dan server berjalan dengan baik.

4.2.3 Perancangan Rangkaian

Rangkaian sistem yang dirancang terdiri atas:

1) Pin I/O 15 ESP32 terhubung ke buzzer *low level trigger*.
2) ESP32 terkoneksi ke jaringan WiFi kampus untuk komunikasi dengan broker MQTT.
3) Catu daya ESP32 diperoleh dari adaptor atau USB 5V.

Catatan implementasi:

1) Akuarium fisik tidak digunakan pada tahap ini.
2) Sistem diuji dalam bentuk simulasi, dengan ketentuan sebagai berikut:
   a) Buzzer berfungsi sebagai simulasi aktuator pemberian pakan.
   b) EEPROM menyimpan jadwal pemberian pakan.
   c) Timer berbasis fungsi millis() digunakan untuk pengaturan waktu.
   d) Aplikasi smartphone atau MQTT *Dashboard* (HiveMQ) digunakan untuk mengatur/*reset* jadwal dan memantau status buzzer.

Dari hasil perancangan, rangkaian dapat berfungsi sesuai dengan logika kontrol yang telah ditentukan.

4.2.4 Pengujian Rangkaian

Pengujian sistem dilakukan dengan skenario sebagai berikut:

1. Jadwal pemberian pakan disimulasikan melalui server MQTT.
2. Saat jadwal tercapai, broker MQTT mengirimkan pesan ke ESP32.
3. ESP32 kemudian memicu buzzer untuk berbunyi sebagai notifikasi.
4. Buzzer berperan sebagai alarm simulasi agar dosen maupun mahasiswa mengetahui waktu pemberian pakan, tanpa memerlukan pemantauan langsung ke akuarium.

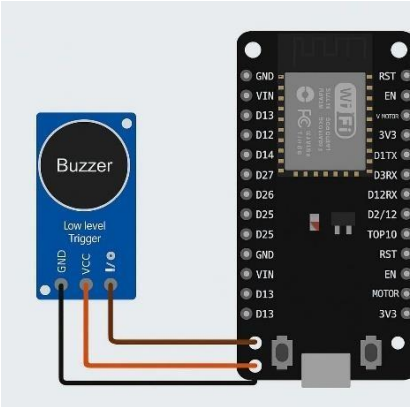
Implementasi simulasi di ruang dosen S1 Teknik Komputer Universitas Dinamika:

a) Perangkat yang digunakan adalah ESP32 DevKit V1 dan *buzzer low level trigger.*
b) Komunikasi dilakukan melalui WiFi kampus dengan broker MQTT (HiveMQ *Dashboard*/*Smartphone App*).

c) Akuarium fisik tidak digunakan, seluruh proses berbasis simulasi jadwal dan status buzzer.

d) Status sistem dapat dipantau secara *real-time* melalui *dashboard* MQTT.

4.3. Rangkaian Skematik Untuk Simulasinya



Gambar 4.3 Rangkaian Skematik

Pada Kerja Praktik dengan judul "Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT di Ruang Dosen S1 Teknik Komputer Universitas Dinamika", rangkaian sistem dirancang dalam bentuk simulasi tanpa menggunakan akuarium fisik. Simulasi ini menggunakan ESP32 DevKit V1 sebagai mikrokontroler utama yang terhubung ke buzzer *low-level trigger*, jaringan WiFi kampus, serta broker MQTT.

4.3.1 Komponen yang Terlibat

A. ESP32 DevKit V1 → Bertugas sebagai pengendali utama, mengatur penjadwalan, manajemen EEPROM, serta komunikasi dengan broker MQTT.

B. Buzzer *Low-Level Trigger* → Berfungsi sebagai simulasi aktuator pemberian pakan, yang akan berbunyi saat pin ESP32 berada pada kondisi logika *LOW*.

C. Kabel Jumper *Male-to-Male* → Digunakan untuk menghubungkan pin I/O ESP32 dengan buzzer.

4.3.2 Sistem Penjadwalan dan Manajemen EEPROM

EEPROM dipakai untuk menyimpan konfigurasi jadwal pemberian pakan, dengan alokasi address sebagai berikut. Tabel address dan Fungsi EEPROM dilihat di bawah ini :

Tabel 4.3.2 Address dan Fungsi EEPROM

Address	Fungsi	Nilai Default	Keterangan
0	Banyaknya bunyi buzzer per hari	2	Dapat diubah melalui aplikasi MQTT
4	Lamanya bunyi buzzer (ms)	500	Durasi buzzer berbunyi
8	*Reset* Banyak	2	Digunakan saat perintah "*Reset* Jadwal"
12	*Reset* LamaMS	500	Digunakan saat perintah "*Reset* Jadwal"

4.3.3 Koneksi WiFi dan MQTT

A. WiFi

1) Koneksi dilakukan pada fungsi `setup()`.
2) Koneksi dicek secara periodik setiap 30 detik. Jika terputus, ESP32 akan melakukan *reconnect* otomatis.

B. MQTT

1) MQTT bergantung pada koneksi WiFi yang aktif.
2) Status koneksi dicek setiap 10 detik.
3) MQTT digunakan untuk *publish* dan *subscribe* dengan topik tertentu, misalnya:
   i. Status buzzer (aktif/mati).
   ii. Perintah "Ubah Jadwal" atau "*Reset* Jadwal".

4.3.4 Aplikasi *Smartphone* IOT MQTT PANEL

1) Informasi yang ditampilkan:
   a) Waktu hitung mundur dalam format jam:menit:detik (2 digit).

b) Status buzzer (aktif/mati).

2) Fitur utama:

a) Ubah Jadwal → *input* jumlah bunyi buzzer (Banyak) dan durasi bunyi (LamaMS) yang kemudian disimpan ke EEPROM.

b) *Reset* Jadwal → mengembalikan konfigurasi jadwal ke nilai default yang tersimpan di EEPROM.

Dengan demikian, rangkaian skematik simulasi ini berhasil membuktikan bahwa integrasi antara ESP32, buzzer, EEPROM, serta komunikasi berbasis MQTT dapat berjalan sesuai dengan logika sistem yang dirancang. Implementasi fisik dengan akuarium dapat dilakukan pada tahap selanjutnya apabila fasilitas tersedia. Sebelum melanjutkan ke tahap pembuatan proyek akhir, terlebih dahulu dilakukan penjelasan secara lebih detail mengenai proses perancangan dan pelaksanaan proyek akhir yang telah dibuat.

4.4 Studi Literatur

Dalam penelitian ini, sistem monitoring notifikasi jadwal pemberian pakan ikan akuarium berbasis IoT menggunakan beberapa komponen utama yang saling terintegrasi. Komponen utama yang digunakan adalah ESP32 yang berfungsi sebagai mikrokontroler sekaligus pusat pengendali. ESP32 memiliki prosesor ganda, dilengkapi modul WiFi dan Bluetooth terintegrasi, serta mendukung berbagai protokol komunikasi sehingga sangat sesuai untuk implementasi *Internet of Things* (IoT).

Selain itu, sistem menggunakan buzzer *Low Level Trigger* sebagai media notifikasi suara. Buzzer akan aktif ketika menerima logika rendah (*LOW*) dan dapat diatur pola bunyinya menggunakan fungsi delay maupun millis agar durasi dan interval notifikasi dapat disesuaikan sesuai kebutuhan.

Untuk mendukung penyimpanan data, digunakan EEPROM yang memungkinkan penyimpanan informasi seperti jadwal pemberian pakan ataupun konfigurasi sistem yang tetap tersimpan meskipun perangkat dimatikan. Sementara itu, koneksi jaringan memanfaatkan WiFi yang sudah tertanam pada ESP32, sehingga perangkat

dapat terhubung ke internet dan melakukan komunikasi data secara *real-time*. Protokol yang digunakan adalah MQTT (*Message Queuing Telemetry Transport*), protokol komunikasi ringan yang banyak digunakan pada aplikasi IoT. MQTT memungkinkan ESP32 mengirim status maupun notifikasi jadwal pemberian pakan ke broker, yang kemudian diteruskan ke perangkat lain seperti komputer atau smartphone untuk keperluan monitoring.

Dengan integrasi ESP32, buzzer, EEPROM, WiFi, dan protokol MQTT, sistem mampu memberikan notifikasi jadwal pemberian pakan ikan akuarium secara efektif dan *real-time*. Aktivasi buzzer diatur menggunakan delay maupun millis sehingga durasi dan interval bunyi dapat disesuaikan. Pengaturan ini terintegrasi dengan penyimpanan data di EEPROM, konektivitas WiFi, serta protokol komunikasi MQTT. Implementasi sistem ini dilakukan sebagai bentuk penerapan konsep IoT dalam lingkungan akademik, khususnya di ruang dosen S1 Teknik Komputer Universitas Dinamika.

4.4.1 Mempelajari EEPROM

Memori non-volatile yang disebut EEPROM (*Electrically Erasable Programmable Read-Only Memory*) memiliki kemampuan untuk menyimpan data meskipun perangkat tidak memiliki pasokan listrik. EEPROM biasanya ada di dalam mikrokontroler, seperti ATmega328 di Arduino UNO, atau dapat dimasukkan ke chip eksternal melalui antarmuka I2C, seperti 24C02 dan 24C256. Meskipun memiliki batas siklus tulis atau hapus (sekitar 100.000 hingga 1.000.000 kali), EEPROM memiliki kelebihan bahwa data dapat ditulis dan dibaca berkali-kali (Arduino, 2023; Banzi & Shiloh, 2014).

Karakteristik utama EEPROM adalah sebagai berikut:

1) Non-volatile → data tetap tersimpan meskipun perangkat dimatikan.
2) Akses per *byte* → memungkinkan pembacaan atau penulisan data pada alamat tertentu.
3) Proses penulisan lebih lambat daripada RAM atau Flash
4) Batas umur pemakaian → tidak cocok untuk aplikasi yang membutuhkan penulisan data terus-menerus (Atmel, 2016).

Fungsi dan Contoh Penggunaan EEPROM:

a) Menyimpan password WiFi ESP32.

b) Menyimpan nilai ambang sensor, misal sensor MQ2 atau suhu.
c) Menyimpan kalibrasi sensor.
d) Mencatat jumlah pakan ikan terakhir (*automatic feeder*).
e) Menyimpan log status relay, motor, atau buzzer (Espressif Systems, 2023).

Cara Penggunaan:

i. Arduino Uno/Nano/Mega → library EEPROM sudah tersedia:

```
#include <EEPROM.h>
```

ii. ESP8266/ESP32 → EEPROM disimulasikan di Flash:

```
EEPROM.begin(size);
EEPROM.commit();
```

Untuk perbandingan singkat terkait Penggunaan EEPROM untuk board Arduino UNO/Nano/Mega dan ESP 8266/ ESP32 dapat dilihat di bawah ini:

Tabel 4.4.1 Perbandingan Singkat Penggunaan EEPROM

Board	**Library**	**Inisialisasi**	**Commit**
Arduino Uno/Nano/Mega	`#include <EEPROM.h>`	Tidak perlu	Tidak perlu
ESP8266/ESP32	`#include <EEPROM.h>`	`EEPROM.begin(size)`	`EEPROM.commit()`

Penulisan dan Pembacaan Data EEPROM:

a) `EEPROM.write(address, value)` → menyimpan 1 *byte.*
b) `EEPROM.put(address, value)` → menyimpan tipe data kompleks *(int, float, struct).*
c) `EEPROM.read(address)` → membaca 1 *byte.*
d) `EEPROM.get(address, value)` → membaca data tipe kompleks.

4.4.2 Mempelajari Koneksi WiFi

ESP32 dan ESP8266 memiliki modul WiFi bawaan, memungkinkan koneksi nirkabel tanpa modul tambahan (Espressif Systems, 2023).

Mode utama WiFi:

A. *Station Mode* (STA) → ESP32 bertindak sebagai klien yang terhubung ke router.

Mengirim/menerima data, akses server, komunikasi antar perangkat.

```
#include <WiFi.h>
const char* ssid = "NAMA_WIFI";
const char* password = "PASSWORD_WIFI";
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.println(WiFi.localIP());
}
void loop() {}
```

B. *Access Point Mode* (AP) → ESP32 membuat hotspot sendiri.

```
WiFi.softAP("ESP32_AP","12345678");
Serial.println(WiFi.softAPIP());
```

C. Mode Ganda (STA + AP) → terhubung ke router dan membuat hotspot sekaligus.

```
WiFi.mode(WIFI_AP_STA);
WiFi.begin(routerSsid, routerPassword);
WiFi.softAP(apSsid, apPassword);
```

Aplikasi setelah terkoneksi**:**

i. Mengirim data ke server/database IoT.

ii. Menjalankan web server.

iii. Menggunakan MQTT untuk komunikasi *real-time.*

iv. *Remote control,* misal menyalakan buzzer pemberitahuan jadwal pakan ikan.

4.4.3 Konsep Dasar MQTT dan Implementasinya pada ESP32/ESP8266

1. Pahami Konsep Dasar

MQTT (*Message Queuing Telemetry Transport*) adalah protokol komunikasi ringan berbasis *publish/subscribe*, yang dirancang untuk aplikasi IoT karena hemat bandwidth dan responsif (Banks & Gupta, 2014). Kapan digunakan: Cocok untuk perangkat IoT seperti ESP32, sensor, dan smart home. Komponen utama:

1. Broker → pusat distribusi pesan, contohnya Mosquitto atau HiveMQ.
2. *Client Publisher* → perangkat yang mengirim data.
3. *Client Subscriber* → perangkat yang menerima data.
4. *Topic* → saluran atau label pesan, misalnya sensor/suhu atau rumah/lampu.

2. *Library* MQTT (Step by Step)

A. *PubSubClient* (Arduino/ESP32/ESP8266)

Library ini paling sering digunakan di Arduino IDE karena ringan dan mudah digunakan, dapat berfungsi sebagai *Publisher* maupun *Subscriber*, serta kompatibel dengan **WiFi.h** (ESP32) atau **ESP8266WiFi.h** (ESP8266) (PubSubClient Library, 2023).

Contoh Kode Dasar:

```
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi
const char* ssid = "NAMA_WIFI";
const char* password = "PASSWORD_WIFI";
```

```
// MQTT Broker
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
WiFiClient espClient;
PubSubClient client(espClient);

// Callback untuk menerima pesan
void callback(char* topic, byte* message, unsigned int
length) {
  Serial.print("Pesan dari topic: ");
  Serial.println(topic);
  Serial.print("Isi: ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
  }
  Serial.println();
}
```

- Fungsi utama: `client.publish()` untuk mengirim pesan, `client.subscribe()` untuk menerima pesan.
- Perlu koneksi WiFi sebelum menggunakan broker (`WiFi.begin(ssid, password)`).

B. *AsyncMqttClient* (ESP32/ESP8266)

Library ini bersifat *non-blocking*, sehingga *loop* ESP32 tidak macet saat menangani banyak *topic* atau menjalankan webserver bersamaan. Cocok untuk proyek *real-time* (ESPAsyncTCP/AsyncTCP) (AsyncMqttClient Library, 2023).

Contoh Kode Singkat:

```
#include <WiFi.h>
#include <AsyncMqttClient.h>

AsyncMqttClient mqttClient;
```

```
void onMqttMessage(char* topic, char* payload,
AsyncMqttClientMessageProperties properties,
                   size_t len, size_t index, size_t total) {
  Serial.print("Pesan dari topic: ");
  Serial.println(topic);
  for (size_t i = 0; i < len; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

`mqttClient.publish()` → mengirim pesan secara asinkron.
Lebih cepat dan stabil untuk menangani banyak topik sekaligus.

3. *Eclipse Mosquitto*

Mosquitto adalah broker MQTT *open-source* yang berfungsi sebagai perantara antara *publisher* dan *subscriber* (Eclipse Mosquitto, 2023).

i. Cara Kerja:
   1. *Publisher* mengirim data ke broker pada topik tertentu.
   2. Broker menerima pesan dan menyimpannya sesuai topik.
   3. *Subscriber* yang *subscribe* ke topik akan menerima pesan tersebut.

ii. Keunggulan *Mosquitto*:
   a. Ringan & cepat, cocok untuk perangkat dengan sumber daya terbatas.
   b. *Open-source* & gratis, banyak digunakan di industri dan penelitian.
   c. Mendukung QoS *(Quality of Service*):
      - QoS 0 → kirim sekali tanpa jaminan diterima.
      - QoS 1 → kirim minimal sekali dengan konfirmasi.
      - QoS 2 → kirim tepat sekali (paling aman).
   d. Dapat dijalankan di Windows, Linux, macOS, bahkan Raspberry Pi.

e. Mendukung autentikasi (*username/password*) dan TLS/SSL untuk keamanan.

4. Contoh Kasus Penerapan MQTT

Misalnya membuat alat monitoring suhu dan asap dengan ESP32:

a) ESP32 (*Publisher*) mengirim data suhu ke *topic* rumah/suhu.
b) ESP32 lain atau aplikasi *smartphone* (*Subscriber*) menerima update *real-time.*
c) Dapat ditambahkan aturan otomatis, misal jika suhu tinggi, buzzer menyala melalui *subscriber*.

4.5 Pengujian Buzzer *Low Level Trigger*

4.5.1 Tujuan Pengujian

Tujuan dari pengujian ini adalah untuk memastikan bahwa modul buzzer aktif rendah (*low level trigger*) yang digunakan pada sistem ESP32 dapat bekerja secara optimal dan sesuai dengan prinsip kerja logika pemicunya. Modul buzzer ini menggunakan sistem *low level trigger*, artinya buzzer akan menyala ketika menerima logika *LOW* (0) dan mati ketika menerima logika *HIGH* (1) dari mikrokontroler. Secara umum, pengujian ini memiliki beberapa tujuan utama sebagai berikut:

1. Memverifikasi fungsi dasar buzzer aktif rendah Pengujian dilakukan untuk memastikan bahwa buzzer dapat berbunyi ketika diberikan logika *LOW* dan berhenti ketika diberikan logika *HIGH*. Hal ini penting karena beberapa jenis buzzer memiliki logika kerja yang berbeda (*high trigger* atau *low trigger*), sehingga verifikasi diperlukan untuk menghindari kesalahan logika kontrol.
2. Menguji pengendalian waktu bunyi dan diam menggunakan fungsi `delay()`Tahap awal pengujian bertujuan memastikan bahwa sistem mampu mengatur durasi buzzer menyala dan mati secara bergantian dengan menggunakan fungsi penundaan sederhana `delay()`. Hal ini digunakan untuk memahami dasar timing dalam kontrol aktuator.

3. Menguji metode kontrol non-blocking menggunakan fungsi `millis()` Pengujian ini dilakukan untuk memastikan sistem dapat mengatur waktu kerja buzzer tanpa menghentikan proses lain pada mikrokontroler. Dengan metode `millis()`, sistem dapat melakukan multitasking seperti membaca sensor, mengirim data, atau menjaga koneksi WiFi/MQTT tanpa terganggu oleh jeda delay.
4. Menguji penyimpanan konfigurasi durasi buzzer menggunakan EEPROM Tujuan selanjutnya adalah memastikan bahwa durasi bunyi buzzer dapat disimpan secara permanen di dalam EEPROM, sehingga meskipun perangkat dimatikan atau di *reset*, nilai konfigurasi tersebut tetap tersimpan dan digunakan kembali pada saat perangkat dinyalakan ulang.
5. Menguji kemampuan sistem dalam mengubah pola bunyi melalui *input* Serial Sistem diuji agar mampu menerima perintah dari pengguna melalui Serial Monitor untuk mengubah pola bunyi (misalnya pola a, b, atau c) dengan variasi waktu *ON* dan *OFF* yang berbeda. Hal ini menunjukkan fleksibilitas sistem dalam pengaturan pola kerja aktuator tanpa perlu memodifikasi kode program.
6. Menguji konektivitas dan kontrol jarak jauh melalui protokol MQTT. Pengujian ini bertujuan memastikan buzzer dapat dikendalikan secara *real-time* melalui jaringan WiFi dan protokol MQTT, dengan topik tertentu untuk *subscribe* dan *publish*. Hal ini membuktikan bahwa sistem buzzer telah terintegrasi dengan konsep IoT *(Internet of Things*) dan dapat menerima perintah dari server atau aplikasi jarak jauh.
7. Menilai stabilitas dan konsistensi kerja buzzer selama operasi jangka panjang Pengujian juga dilakukan untuk mengamati apakah buzzer tetap dapat beroperasi stabil dalam jangka waktu tertentu, tanpa mengalami delay yang tidak diinginkan atau kesalahan logika akibat penggunaan metode waktu yang berbeda.

   Dengan demikian, secara keseluruhan, pengujian ini tidak hanya berfokus pada fungsi dasar buzzer, tetapi juga menilai keandalan, fleksibilitas, dan integrasi sistem kontrol berbasis ESP32 baik secara lokal maupun melalui jaringan IoT. Hasil pengujian ini akan menjadi dasar dalam menentukan

keefektifan rangkaian dan program dalam implementasi sistem notifikasi berbasis buzzer.

4.5.2 Alat dan Bahan

Pada tahap pengujian buzzer *low level trigger* ini, digunakan beberapa alat dan bahan untuk mendukung proses perancangan, pemrograman, serta pengujian sistem berbasis ESP32. Adapun alat dan bahan yang digunakan dijelaskan sebagai berikut:

1. Alat

Alat yang digunakan dalam pengujian berfungsi untuk membantu proses perakitan, pemrograman, dan pemantauan hasil pengujian sistem buzzer. Berikut daftar alat yang digunakan bisa dilihat Pada tabel di bawah ini:

Tabel 4.5.2 Nama Alat dan Fungsi

No	Nama Alat	Fungsi
1	Laptop	Digunakan untuk menulis kode program, melakukan kompilasi, serta mengunggah program ke mikrokontroler ESP32 menggunakan software Arduino IDE.
2	Kabel Data USB Micro USB	Sebagai media penghubung antara laptop dan board ESP32 untuk proses upload program dan komunikasi serial.
3	Kabel Jumper (Male to Female) (Male to Male) (Female to Female)	Digunakan untuk menghubungkan pin ESP32 dengan pin pada modul buzzer secara fleksibel.
4	Koneksi WiFi Lokal	Diperlukan untuk melakukan pengujian sistem berbasis IoT yang melibatkan koneksi MQTT secara *real-time*.

2. Bahan

Bahan yang digunakan terdiri atas perangkat keras dan perangkat lunak pendukung sistem. Setiap bahan memiliki peran penting dalam pengujian fungsi buzzer berbasis ESP32. Berikut daftar bahan yang digunakan bisa dilihat Pada tabel di bawah ini:

Tabel 4.5.2 Nama Bahan, Spesifikasi, dan Fungsi

No	Nama Bahan	Spesifikasi	Fungsi
1	ESP32 DoIt DevKit V1	Mikrokontroler dengan prosesor dual-core 240 MHz, Wi-Fi 802.11 b/g/n, Bluetooth v4.2, dan GPIO multifungsi	Berfungsi sebagai pusat kendali sistem yang mengatur logika kerja buzzer dan komunikasi jaringan.
2	Modul Buzzer Aktif (*Low Level Trigger*)	Tegangan kerja 3.3V – 5V	Sebagai aktuator yang menghasilkan bunyi saat menerima logika *LOW* dari ESP32.
3	Software Arduino IDE	Versi 2.x atau terbaru	Sebagai platform pemrograman untuk menulis, mengunggah, dan memantau hasil program melalui Serial Monitor.
4	*Library* Tambahan		
• EEPROM.h	-	Untuk menyimpan dan membaca data konfigurasi buzzer	

		(misalnya durasi bunyi) secara permanen di memori internal ESP32.	
• WiFi.h	-	Untuk menghubungkan ESP32 ke jaringan WiFi lokal dalam komunikasi IoT.	
• PubSubClient.h	-	Untuk mengimplementasikan komunikasi MQTT antara ESP32 dan broker IoT.	
5	Broker MQTT: broker.hivemq.com Untuk pengujian Buzzer *Low Level Trigger* Atau mqtt.dinamika.ac.id Untuk project akhir	*Public* MQTT broker	Digunakan sebagai perantara (server) untuk komunikasi pesan antara ESP32 dan perangkat atau aplikasi lain melalui protokol MQTT.

4.5.3 Deskripsi Singkat Komponen Utama

Pada pengujian sistem buzzer berbasis ESP32 DoIt DevKit V1, terdapat beberapa komponen utama yang saling terintegrasi untuk membentuk sistem kontrol berbasis *Internet of Things* (IoT). Masing-masing komponen memiliki peran penting dalam memastikan sistem dapat berfungsi secara optimal. Berikut adalah penjelasan dari setiap komponen utama:

a. ESP32 DoIt DevKit V1

ESP32 merupakan mikrokontroler generasi lanjutan yang dikembangkan oleh Espressif Systems, dan menjadi penerus dari seri ESP8266 dengan

kemampuan yang lebih tinggi. Menurut Mischianti (n.d.), ESP32 dilengkapi dengan prosesor *dual-core* Tensilica LX6 berkecepatan hingga 240 MHz, memori SRAM internal, serta dukungan konektivitas Wi-Fi 802.11 b/g/n dan Bluetooth v4.2 (*Classic* dan BLE).

Dalam pengujian ini, ESP32 berfungsi sebagai pusat pengendali utama yang:

1. Mengatur kondisi *ON/OFF* buzzer berdasarkan sinyal logika digital (*LOW/HIGH*).
2. Menyimpan konfigurasi durasi bunyi buzzer di EEPROM, agar data tetap tersimpan meskipun perangkat di *reset*.
3. Mengatur komunikasi jaringan menggunakan protokol Wi-Fi dan MQTT, untuk memungkinkan kontrol jarak jauh melalui broker HiveMQ.

Selain itu, ESP32 dipilih karena memiliki kemampuan pemrosesan yang cepat, port *input/output* (GPIO) yang fleksibel, serta kompatibilitas tinggi dengan Arduino IDE, sehingga mempermudah proses pemrograman dan integrasi perangkat.

b. Modul Buzzer Aktif Low (*Low Level Trigger*)

Modul buzzer aktif low merupakan aktuator audio yang menghasilkan suara ketika menerima logika digital *LOW* (0V**)** dan berhenti berbunyi ketika menerima logika *HIGH* (3.3V). Jenis buzzer ini disebut *low level trigger*, karena aktif saat tegangan rendah.

Dalam sistem ini:

a) Ketika ESP32 memberikan sinyal *LOW* → buzzer *ON* (berbunyi).
b) Ketika ESP32 memberikan sinyal *HIGH* → buzzer *OFF* (diam).

Karena ESP32 beroperasi pada tegangan logika 3.3V, buzzer tipe ini sangat cocok digunakan karena tidak membutuhkan arus besar dan tetap dapat bekerja stabil tanpa transistor tambahan. Buzzer ini berfungsi sebagai indikator suara dalam sistem notifikasi, yang digunakan untuk memberi tanda atau peringatan berdasarkan program yang dijalankan.

c. Koneksi Wi-Fi Lokal dan Broker MQTT (HiveMQ)

Dalam pengujian ini, ESP32 dihubungkan ke jaringan Wi-Fi lokal agar dapat berkomunikasi secara nirkabel dengan broker MQTT, yaitu broker publik HiveMQ yang beralamat di broker.hivemq.com, untuk pengujian sebelum tahap ke project akhir,

Broker MQTT berfungsi sebagai server komunikasi IoT tempat ESP32 mengirim dan menerima pesan melalui topik tertentu.

a) Topik *Subscribe***:** esp32/buzzer — digunakan untuk menerima perintah *ON* atau *OFF* dari pengguna atau aplikasi jarak jauh.
b) Topik *Publish***:** esp32/status — digunakan untuk mengirim status buzzer (aktif/mati) ke server.

Dengan sistem ini, ESP32 dapat dikontrol secara *real-time* melalui internet, sehingga konsep *Internet of Things* (IoT) benar-benar diterapkan pada pengujian buzzer low level trigger ini.

4.5.4 Skema Hubungan Fisik Rangkaian

Pada pengujian ini, komponen ESP32 dan buzzer dihubungkan dalam rangkaian sederhana dengan konfigurasi kabel sebagai berikut ada di tabel bawah ini.

Tabel 4.5.4 Komponen Buzzer, Terhubung ke ESP32, dan Fungsi

Komponen Buzzer	**Terhubung ke ESP32**	**Fungsi**
VCC	3.3V	Sebagai sumber tegangan utama untuk mengaktifkan modul buzzer.
GND	GND	Jalur ground untuk melengkapi rangkaian arus listrik.
IN (*Input* Sinyal)	GPIO 15	Jalur kontrol sinyal digital dari ESP32 untuk menyalakan atau mematikan buzzer.

Penjelasan Pemilihan GPIO 15

Pin GPIO 15 dipilih karena merupakan pin *output* yang aman dan stabil digunakan untuk aplikasi kontrol digital. Pin ini tidak memengaruhi proses booting ESP32 dan dapat menghasilkan sinyal logika *HIGH/LOW* dengan cepat dan konsisten. Selain itu, GPIO 15 mendukung arus keluaran yang cukup untuk mengaktifkan buzzer aktif tanpa perlu menggunakan transistor tambahan.

Prinsip Kerja Rangkaian

1. Ketika ESP32 memberikan sinyal *LOW* (0V) pada pin GPIO 15, maka buzzer aktif dan menghasilkan bunyi.
2. Ketika sinyal berubah menjadi *HIGH* (3.3V), buzzer akan mati.
3. Program pada ESP32 mengatur durasi bunyi dan diam dengan fungsi `delay()` atau `millis()`.
4. Konfigurasi durasi dapat disimpan di EEPROM dan diubah melalui Serial Monitor atau perintah jarak jauh lewat MQTT.

4.5.5 Langkah Pengujian

Pengujian dilakukan melalui beberapa skenario program untuk memastikan bahwa buzzer aktif *low* bekerja sesuai logika pemicu yang diinginkan, serta dapat merespons dengan tepat terhadap sinyal digital dari ESP32 DoIt DevKit V1. Tahap awal dilakukan pengujian dasar untuk memastikan fungsi utama buzzer, yaitu bekerja pada logika *LOW* (menyala) dan berhenti pada logika *HIGH* (mati).

4.5.6 Pengujian Buzzer 1 Detik *ON* – 1 Detik *OFF*

Pada skenario pertama ini, buzzer diuji dengan menyalakan dan mematikannya secara bergantian setiap 1 detik, menggunakan fungsi `delay()` pada mikrokontroler ESP32. Tujuan pengujian ini adalah untuk memverifikasi respon buzzer terhadap logika *HIGH* dan *LOW* serta memastikan bahwa pin GPIO 15 berfungsi dengan baik sebagai *output* kontrol.

Kode Program Pengujian Dasar:

```
#define BUZZER_PIN 15   // ganti dengan pin yang dipakai


void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH); // buzzer mati (karena
low trigger)
}


void loop() {
  digitalWrite(BUZZER_PIN, LOW);   // buzzer ON
  delay(1000);                     // tunggu 1 detik
  digitalWrite(BUZZER_PIN, HIGH);  // buzzer OFF
  delay(1000);                     // tunggu 1 detik
}
```

Penjelasan Program Pengujian Buzzer Aktif *Low*

Program ini digunakan untuk mengaktifkan dan menonaktifkan buzzer secara bergantian dengan interval waktu 1 detik menggunakan fungsi `delay()`. Buzzer yang digunakan bertipe aktif low (low level trigger), artinya buzzer akan menyala saat menerima logika *LOW* (0V) dan mati saat menerima logika *HIGH* (3.3V) dari mikrokontroler ESP32.

1. Bagian Inisialisasi Pin

Pada bagian awal program terdapat perintah:

```
#define BUZZER_PIN 15
```

Baris ini mendefinisikan pin GPIO 15 sebagai jalur yang digunakan untuk mengontrol buzzer. Dengan cara ini, pemrograman menjadi lebih mudah karena nama BUZZER_PIN bisa langsung dipanggil di seluruh program tanpa perlu menulis angka pin secara berulang. GPIO 15 dipilih karena merupakan pin aman untuk *output* dan tidak mengganggu proses booting ESP32.

2. Bagian Setup

Bagian `setup()` dijalankan satu kali saat perangkat pertama kali dinyalakan. Perintah pinMode(`BUZZER_PIN, OUTPUT`); mengatur pin GPIO 15 menjadi pin keluaran, sehingga ESP32 dapat memberikan sinyal logika *HIGH* (3.3V) atau *LOW* (0V) ke buzzer. Kemudian, perintah digitalWrite(BUZZER_PIN, HIGH); mengirimkan logika *HIGH* ke pin tersebut. Karena buzzer bertipe aktif *low*, maka logika *HIGH* akan membuat buzzer mati atau diam. Tujuannya agar saat ESP32 dinyalakan pertama kali, buzzer tidak langsung berbunyi.

3. Bagian Loop

Fungsi `loop()` akan berjalan berulang-ulang tanpa henti selama perangkat menyala. Pada bagian ini, buzzer dikendalikan secara bergantian antara kondisi *ON dan OFF* menggunakan logika digital dan waktu tunda (`delay()`).

- `digitalWrite(BUZZER_PIN, LOW);` memberikan logika *LOW* (0V) ke pin, sehingga buzzer menyala (berbunyi).
- `delay(1000);` memberikan jeda waktu 1 detik (1000 milidetik) sebelum instruksi berikutnya dijalankan.
- `digitalWrite(BUZZER_PIN, HIGH);` memberikan logika HIGH (3.3V) ke pin, sehingga buzzer mati (diam).
- `delay(1000);` memberi jeda 1 detik lagi sebelum siklus pengulangan dimulai kembali.

Dengan demikian, buzzer akan berbunyi selama 1 detik, lalu diam 1 detik, secara terus-menerus.

4. Prinsip Kerja Program

Program ini memanfaatkan fungsi `delay()` untuk mengatur waktu nyala dan mati buzzer. Meskipun cara ini sederhana dan efektif untuk pengujian dasar, metode ini bersifat blocking, artinya selama `delay()` berjalan, mikrokontroler tidak dapat menjalankan proses lain. Namun, untuk uji awal seperti memastikan logika kerja buzzer, metode ini sudah cukup efektif dan mudah dipahami.

Kesimpulan Penjelasan

1) Tujuan program**:** menguji respon buzzer terhadap sinyal logika *HIGH* dan *LOW.*
2) Hasil: buzzer berbunyi saat logika *LOW* dan diam saat logika *HIGH.*
3) Interval kerja: 1 detik bunyi dan 1 detik diam.
4) Pin kontrol: GPIO 15, aman digunakan pada ESP32.
5) Metode waktu: menggunakan fungsi delay() dengan nilai 1000 milidetik.

4.5.7 Pengujian Buzzer Menyala Setiap 10 Detik Sekali

1. Tujuan Pengujian

Tujuan dari pengujian ini adalah untuk memastikan buzzer aktif *low* dapat diatur menyala secara periodik dengan interval waktu tertentu, yaitu berbunyi selama 0,5 detik setiap 10 detik sekali. Pengujian ini juga berfungsi untuk menguji akurasi fungsi waktu `delay()` dalam menghasilkan periode kerja yang stabil, serta memverifikasi kestabilan logika digital dari pin GPIO 15 pada ESP32 sebagai pin kendali buzzer.

2. Kode Program Pengujian

```
#define BUZZER_PIN 15   // pin buzzer



void setup() {
```

```
        pinMode(BUZZER_PIN, OUTPUT);
        digitalWrite(BUZZER_PIN, HIGH); // buzzer OFF (karena low trigger)
      }
void loop() {
        // 🔊 Hidupkan buzzer selama 0.5 detik
        digitalWrite(BUZZER_PIN, LOW); // buzzer ON
        delay(500);                    // 500 ms = 0.5 detik

        // 🔇 Matikan buzzer
        digitalWrite(BUZZER_PIN, HIGH); // buzzer OFF

        // Tunggu 9.5 detik agar total siklus = 10 detik
        delay(9500);
      }
```

3. Penjelasan Program

i. Inisialisasi Pin

```
#define BUZZER_PIN 15
```

Mendefinisikan pin GPIO 15 sebagai jalur kontrol buzzer. Jika posisi pin buzzer diubah, pengguna cukup mengganti angka ini tanpa perlu mengubah logika program.

```
pinMode(BUZZER_PIN, OUTPUT);
```

Mengatur pin GPIO 15 sebagai *output*, agar ESP32 dapat memberikan sinyal digital *HIGH/LOW* ke modul buzzer.

`digitalWrite(BUZZER_PIN, HIGH);`

Memberikan logika HIGH (3.3V) ke pin buzzer sehingga buzzer dalam kondisi mati, karena modul buzzer ini bertipe *low level trigger*.

✓ Karakteristik modul buzzer aktif low:

a) LOW (0V) → Buzzer menyala (berbunyi).

b) *HIGH* (3.3V) → Buzzer mati (diam).

ii. Logika Kerja Utama (Loop Program)

a) `digitalWrite(BUZZER_PIN, LOW);`
   Mengirim sinyal *LOW* ke buzzer → buzzer hidup.

b) `delay(500);`
   Memberikan jeda waktu 500 milidetik (0,5 detik) saat buzzer menyala.

c) `digitalWrite(BUZZER_PIN, HIGH);`
   Mengirim sinyal *HIGH* → buzzer mati.

d) `delay(9500);`
   Memberikan jeda waktu 9,5 detik sebelum siklus diulang.

Dengan pengaturan waktu di atas, total satu siklus kerja buzzer menjadi 10 detik:

1) Bunyi 0,5 detik,
2) Diam 9,5 detik,
3) Lalu berulang secara terus-menerus.

iii. Analisis Waktu dan Sinyal

1) Periode Siklus (**T**)

Periode merupakan waktu yang dibutuhkan untuk satu kali siklus lengkap, yaitu saat buzzer menyala (aktif) dan mati (diam). Rumus periode dinyatakan sebagai:

$$T = tbunyi + tdiam$$

Dengan:

- $tbunyi$=0,5 detik
- $tdiam$=9,5 detik

Maka:

$$T = 0,5 + 9,5 = 10\,\text{detik}$$

Artinya, buzzer berbunyi setiap 10 detik sekali dalam satu siklus penuh. Menurut Zenius (2021), periode merupakan waktu yang dibutuhkan suatu getaran atau gelombang untuk menyelesaikan satu siklus penuh sebelum kembali ke posisi semula.

2) Frekuensi (**f**)

Hubungan antara frekuensi dan periode mengikuti persamaan dasar fisika gelombang:

$$f = \frac{1}{T}$$

Dengan **T**=10 detik, diperoleh:

$$\mathbf{f = \frac{1}{10} = 0,1\ Hz}$$

Artinya, buzzer berbunyi 0,1 kali per detik, atau 1 kali setiap 10 detik. Sejalan dengan penjelasan Zenius (2021), frekuensi menunjukkan jumlah getaran atau siklus yang terjadi setiap satu detik, dan satuannya adalah ***Hertz*** (**Hz**).

3) *Duty Cycle* (**D**)

Duty cycle menunjukkan persentase waktu aktif (bunyi) dibandingkan waktu total siklus.

Rumusnya:

$$D = \frac{tbunyi}{T} \times 100\%$$

Dengan nilai:

$$D = \frac{0,5}{10} \times 100\%$$

Artinya, buzzer hanya aktif selama 5% dari total waktu siklus, dan 95% sisanya dalam keadaan mati. *Duty cycle* ini penting untuk mengatur efisiensi energi serta durasi notifikasi bunyi agar tidak terlalu lama.

- Pola Kerja Sinyal Buzzer

Berdasarkan hasil pengujian program:

a) □ Bunyi selama 0,5 detik (logika *LOW*)

b) ⊗ Diam selama 9,5 detik (logika *HIGH*)

c) a Ulang terus menerus setiap 10 detik

Pola tersebut menunjukkan bahwa program telah bekerja sesuai dengan konsep sinyal periodik dan perhitungan waktu yang telah dirumuskan.

4.5.8 Pengujian Buzzer Bunyi Setiap 1 Menit Selama 5 Detik

Kode Program :

```
#define BUZZER_PIN 15   // pin buzzer (low trigger)

// Variabel konfigurasi (bisa diubah sesuai kebutuhan)
unsigned long tBunyi = 5000;       // lama bunyi (ms) ->
contoh: 5 detik = 5000
unsigned long tPeriod = 60000;     // periode total (ms) -
> contoh: 1 menit = 60000

void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH); // buzzer mati saat
mulai (active-low)
}

void loop() {
  // Buzzer ON
  digitalWrite(BUZZER_PIN, LOW);
  delay(tBunyi);                   // tunggu selama bunyi

  // Buzzer OFF
  digitalWrite(BUZZER_PIN, HIGH);
  delay(tPeriod - tBunyi);         // tunggu sisa waktu
agar total = tPeriod
}
```

- Penjelasan Program

1) Definisi Pin

```
#define BUZZER_PIN 15
```

Baris ini mendefinisikan pin GPIO 15 sebagai jalur kontrol buzzer. Jika pin diganti, cukup ubah angka 15 sesuai dengan pin yang digunakan pada ESP32. Pin ini dipilih karena termasuk pin *output* aman yang tidak mengganggu proses *booting* mikrokontroler.

2) Variabel Konfigurasi

```
unsigned long tBunyi = 5000;       // lama bunyi (ms)
unsigned long tPeriod = 60000;     // periode total (ms)
```

- $tBunyi$ = lama buzzer menyala dalam milidetik (5000 ms = 5 detik).
- $tPeriod$ = periode total antar bunyi (60000 ms = 1 menit).

Waktu buzzer dalam keadaan diam (mati) dihitung dengan rumus:

$$tdiam = tperiod - tbunyi$$

3) Bagian Setup

```
pinMode(BUZZER_PIN, OUTPUT);
digitalWrite(BUZZER_PIN, HIGH);
```

- `pinMode(BUZZER_PIN, OUTPUT)` → mengatur GPIO 15 menjadi *output* digital.
- `digitalWrite(BUZZER_PIN, HIGH)` → mengirim logika *HIGH* (3.3V) agar buzzer mati saat awal.

➢ Karena buzzer bertipe *active-low*, maka:

a) *LOW* (0V) → Buzzer hidup (berbunyi).

b) *HIGH* (3.3V) → Buzzer mati (diam).

4) Bagian Loop

```
digitalWrite(BUZZER_PIN, LOW);   // buzzer ON
delay(tBunyi);                   // tunggu selama bunyi
digitalWrite(BUZZER_PIN, HIGH);  // buzzer OFF
delay(tPeriod - tBunyi);         // tunggu sisa waktu
```

✓ Saat logika *LOW* diberikan, buzzer aktif (berbunyi) selama $tBunyi$ milidetik.

✓ Setelah itu, buzzer dimatikan dengan logika *HIGH*, lalu menunggu sisa waktu (tPeriod - tBunyi) agar total waktu per siklus tetap 1 menit.

5) Perhitungan dan Analisis

Misalnya:

✓ $tbunyi$=5000ms=5detik

- ✓ $tperiod$=60000ms=60detik=1menit
- ✓ Maka:

$$tdiam = tperiod - tbunyi = 60000 - 5000 = 55000\,ms = 55\,detik$$

✓ Artinya:

a) Buzzer *ON* selama 5 detik.
b) Buzzer *OFF* selama 55 detik.
c) Pola berulang setiap 1 menit.

Hubungan antara **frekuensi (f)** dan **periode (T)** mengikuti persamaan:

$$f = \frac{1}{T}$$

(Zenius, 2021)

Dengan $T = 60\,detik$, maka:

$$f = \frac{1}{60} = 0,0167Hz$$

Frekuensi ini berarti buzzer menyala sekali setiap 60 detik.
Menurut teori gelombang periodik (Zenius, 2021), semakin besar nilai periode, maka frekuensi bunyi menjadi semakin kecil.

✓ Pola Kerja Sistem

a) □ Buzzer berbunyi selama 5 detik.
b) 🔇 Buzzer diam selama 55 detik.
c) a Pola ini diulang terus-menerus setiap 1 menit.

Dengan logika ini, sistem menghasilkan sinyal periodik yang stabil dan presisi, sesuai teori dasar hubungan antara periode dan frekuensi.

4.5.9 Kodingan Buzzer Bunyi Menggunakan `Millis()`

Kode Program:

```
#define BUZZER_PIN 15   // pin buzzer (low trigger)


// Variabel konfigurasi (bisa diubah sesuai kebutuhan)
unsigned long tBunyi = 5000;       // lama bunyi (ms) -> 5 detik = 5000
unsigned long tPeriod = 60000;     // periode total (ms) -> 1 menit = 60000


// Variabel internal
unsigned long prevMillis = 0;      // waktu terakhir buzzer ON/OFF
bool buzzerState = false;          // kondisi buzzer (ON/OFF)


void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH);  // buzzer OFF saat mulai (active-low)
}


void loop() {
  unsigned long currentMillis = millis();
  unsigned long elapsed = currentMillis - prevMillis;


  if (buzzerState) {
    // Kalau buzzer sedang ON
    if (elapsed >= tBunyi) {
      digitalWrite(BUZZER_PIN, HIGH);   // matikan buzzer
      buzzerState = false;
```

```
      prevMillis = currentMillis;       // reset timer
    }
  } else {
    // Kalau buzzer sedang OFF
    if (elapsed >= (tPeriod - tBunyi)) {
      digitalWrite(BUZZER_PIN, LOW);    // hidupkan buzzer
      buzzerState = true;
      prevMillis = currentMillis;       // reset timer
    }
  }
}
```

✓ Penjelasan Program

1) Definisi Pin

```
#define BUZZER_PIN 15
```

Menetapkan buzzer pada pin GPIO 15. Penggunaan `#define` membuat kode lebih fleksibel — jika pin diganti, cukup ubah angka tanpa memodifikasi bagian lain program. Pin GPIO 15 dipilih karena aman digunakan sebagai *output* digital pada ESP32 tanpa mengganggu proses *booting*.

2) Variabel Konfigurasi

```
unsigned long tBunyi = 5000;
unsigned long tPeriod = 60000;
```

✓ $tBunyi$ = durasi buzzer hidup dalam milidetik (5000 ms = 5 detik).

✓ $tPeriod$ = waktu total satu siklus (60000 ms = 1 menit).

Waktu buzzer dalam keadaan **mati (diam)** dihitung menggunakan rumus:

$$tdiam = tperiod - tbunyi$$

(Sumber: Zenius, 2021)

Dengan nilai tersebut:

$$tdiam = 60000 - 5000 = 55000ms = 55detik$$

Artinya buzzer akan *ON* selama 5 detik dan *OFF* selama 55 detik, berulang setiap 1 menit.

3) Variabel Internal

```
unsigned long prevMillis = 0;
bool buzzerState = false;
```

a) `prevMillis` → menyimpan waktu terakhir perubahan status buzzer.

b) `buzzerState .` menandakan kondisi buzzer saat ini (*true = ON, false = OFF*).

4) Setup Awal

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

✓ `pinMode()` mengatur pin buzzer sebagai *output*.

✓ `digitalWrite(HIGH)` memastikan buzzer mati saat awal.

Karena buzzer bertipe *active-low*, maka:

a) *LOW* (0V) → buzzer menyala (berbunyi).

b) *HIGH* (3.3V) → buzzer mati (diam).

5) Loop Utama

Program utama tidak menggunakan `delay()` melainkan fungsi `millis()`, sehingga sistem tetap responsif terhadap proses lain seperti komunikasi sensor atau Wi-Fi.

Logika Program**:**

a) Menghitung waktu yang telah berlalu (elapsed) sejak perubahan status terakhir.

b) Jika buzzer *ON* selama ≥ $tBunyi$, maka buzzer dimatikan.

c) Jika buzzer *OFF* selama ≥ $(tPeriod - tBunyi)$, maka buzzer dinyalakan kembali.

Dengan demikian, satu siklus total tetap 1 menit, tetapi tanpa menghentikan program utama.

✓ Alur Kerja Sistem

i. Buzzer awalnya mati (*OFF*).
ii. Setelah 55 detik, buzzer menyala (*ON*) selama 5 detik.
iii. Setelah itu, buzzer mati kembali dan siklus diulang terus menerus.
iv. Seluruh waktu dihitung menggunakan `millis()` secara non-blocking.

➢ Keunggulan Penggunaan `millis()`

a) *Non-blocking*: ESP32 tetap bisa menjalankan tugas lain sambil menunggu waktu.
b) Lebih efisien: Tidak menghentikan seluruh sistem seperti `delay()`.
c) Profesional: Digunakan dalam sistem nyata seperti IoT dan kontrol industri.

Menurut teori fisika tentang periode dan frekuensi (Zenius, 2021), hubungan antara waktu aktif dan waktu total mengikuti:

$$f = \frac{1}{T}$$

Dengan $T = 60detik$, maka frekuensi sinyal buzzer:

$$f = 0,0167Hz$$

yang berarti 1 kali bunyi setiap 60 detik.

➢ Analisis Waktu dan *Duty Cycle*

Dari hasil pengujian:

- $tbunyi = 5detik$
- $tdiam = 55detik$
- $T = 60detik$

Duty cycle dihitung sebagai:

$$D = \frac{tbunyi}{T} \times 100\% = \frac{5}{60} \times 100\% = 8,33\%$$

Artinya buzzer hanya aktif 8,33% dari waktu total siklus. Perhitungan ini sesuai dengan prinsip periode dan frekuensi gelombang periodik sebagaimana dijelaskan dalam *Zenius (2021)* bahwa semakin besar periode sinyal, maka frekuensi bunyinya semakin kecil.

✓ Kesimpulan Pengujian

Program ini berhasil membuat buzzer menyala 5 detik setiap 1 menit sekali menggunakan metode *non-blocking* berbasis `millis()`. Seluruh logika waktu bekerja stabil dengan siklus tetap 60 detik tanpa gangguan terhadap fungsi mikrokontroler lainnya.

4.5.10 Pengujian Buzzer Menggunakan EEPROM

A. Kodingan EEPROM pada buzzer *Low level trigger* :

```
#include <EEPROM.h>

#define BUZZER_PIN 15   // pin buzzer (LOW trigger)
#define EEPROM_SIZE 512 // ukuran EEPROM (wajib di-
EEPROM.begin())

int addr = 0;           // alamat EEPROM untuk simpan
durasi
unsigned long durasi;   // variabel durasi buzzer (ms)

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

```
  pinMode(BUZZER_PIN, OUTPUT);

  // Inisialisasi EEPROM
    if (!EEPROM.begin(EEPROM_SIZE)) {
      Serial.println("Gagal inisialisasi EEPROM!");
      while (1);
    }

    // --- Membaca data lama dari EEPROM ---
    EEPROM.get(addr, durasi);

    // Jika masih kosong (belum pernah disimpan), set
default
    if (durasi == 0xFFFFFFFF || durasi == 0) {
      durasi = 1000;   // default 1 detik (1000 ms)
      EEPROM.put(addr, durasi);
      EEPROM.commit();
    }

    Serial.print("Durasi dari EEPROM: ");
    Serial.println(durasi);
  }
```

```
void loop() {
  // Bunyi buzzer pakai durasi yang dibaca dari
EEPROM
  digitalWrite(BUZZER_PIN, LOW);   // aktif (karena
low trigger)
  delay(durasi);
  digitalWrite(BUZZER_PIN, HIGH); // mati
  delay(500);   // jeda setengah detik

  // --- Update durasi via Serial Monitor ---
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    unsigned long durasiBaru = input.toInt();
    if (durasiBaru > 0) {
      durasi = durasiBaru;
      EEPROM.put(addr, durasi);
      EEPROM.commit();
      Serial.print("Durasi baru tersimpan: ");
      Serial.println(durasi);
    }
  }
}
```

- Penjelasan Program

Program ini bertujuan untuk mengontrol durasi bunyi buzzer menggunakan EEPROM, sehingga nilai durasi dapat disimpan secara permanen di dalam memori mikrokontroler dan tetap tersedia meskipun daya dimatikan.

A. Inisialisasi Awal

Bagian awal program memanggil pustaka EEPROM dan mendefinisikan parameter dasar:

i. `#include <EEPROM.h>` → mengaktifkan fungsi baca-tulis pada memori EEPROM internal ESP32.

ii. `#define BUZZER_PIN 15 .` menetapkan GPIO 15 sebagai pin *output* buzzer bertipe low-level trigger, artinya buzzer menyala saat logika bernilai *LOW*.

iii. `#define EEPROM_SIZE 512 .` menentukan ukuran memori EEPROM yang akan digunakan (512 *byte*).

iv. `int addr = 0 .` menentukan alamat awal penyimpanan data durasi.

v. `unsigned long durasi .` menyimpan nilai durasi bunyi buzzer dalam satuan milidetik.

Fungsi `EEPROM.begin(EEPROM_SIZE)` wajib dipanggil agar ESP32 dapat mengakses memori EEPROM-nya.

B. Penetapan Nilai Default Durasi

Saat pertama kali program dijalankan, sistem akan mencoba membaca data yang sebelumnya tersimpan di EEPROM menggunakan:

```
EEPROM.get(addr, durasi);
```

ketika data belum pernah disimpan (hasilnya kosong atau 0), maka nilai *default* durasi = 1000 ms (1 detik) akan ditetapkan dan disimpan kembali menggunakan:

```
EEPROM.put(addr, durasi);
EEPROM.commit();
```

Perintah `EEPROM.commit()` sangat penting karena memastikan data benar-benar tertulis ke memori flash ESP32.

C. Membaca Data dari EEPROM

Nilai durasi yang tersimpan di EEPROM akan ditampilkan melalui Serial Monitor, dengan pesan seperti:

```
Durasi dari EEPROM: 1000
```

Dan Pada gambar berikut dibawah ini

```
12:17:58.950 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
12:17:58.950 -> mode:DIO, clock div:1
12:17:58.950 -> load:0x3fff0030,len:4744
12:17:58.993 -> load:0x40078000,len:15672
12:17:58.993 -> load:0x40080400,len:3152
12:17:58.993 -> entry 0x4008059c
12:17:59.121 -> Durasi dari EEPROM: 1000
```

Gambar 4.5.5.5 *Output* Durasi EEPROM

Pada Gambar ini menandakan bahwa sistem berhasil membaca nilai durasi dari memori *non-volatile*.

D. Bagian Loop (Logika Utama)

Bagian `loop()` menjalankan dua fungsi utama:

1. Mengaktifkan buzzer berdasarkan durasi tersimpan.
   i. `digitalWrite(BUZZER_PIN, LOW)` → buzzer *ON* (karena *low trigger*).
   ii. `delay(durasi)` → menunggu selama nilai durasi aktif (misal 1000 ms).
   iii. `digitalWrite(BUZZER_PIN, HIGH)` → buzzer *OFF*.
   iv. `delay(500)` → jeda 0,5 detik sebelum siklus berikutnya.
2. Memperbarui durasi melalui Serial Monitor.

   Jika pengguna mengetik angka baru (misal 3000) di Serial Monitor, maka:
   i. Nilai durasi akan diperbarui menjadi 3000 ms.
   ii. Data baru disimpan ke EEPROM dengan `EEPROM.put()` dan `EEPROM.commit()`.
   iii. Serial Monitor akan menampilkan:

   ```
   Durasi baru tersimpan: 3000
   ```

Dengan demikian, setiap kali ESP32 dihidupkan ulang, durasi terakhir yang disimpan tetap digunakan tanpa harus diatur ulang.

- Alur Kerja Sistem

1. Sistem membaca nilai durasi dari EEPROM.
2. Jika belum ada data, sistem menetapkan nilai *default* (1 detik).
3. Buzzer menyala sesuai durasi yang tersimpan.
4. Pengguna dapat mengirim nilai baru melalui Serial Monitor untuk memperbarui durasi.
5. Nilai baru otomatis disimpan ke EEPROM dan digunakan pada siklus berikutnya.

- Analisis Teknis

Penggunaan EEPROM pada ESP32 memungkinkan sistem menyimpan konfigurasi secara permanen, yang sangat penting pada aplikasi IoT atau sistem tertanam (*embedded system*).

Dibandingkan metode hardcode biasa, pendekatan ini:

i. Lebih fleksibel, karena parameter dapat diubah tanpa memodifikasi program.
ii. *Non-volatile*, artinya data tidak hilang saat perangkat dimatikan.
iii. Efisien untuk kalibrasi perangkat dan pengaturan waktu otomatis.

Hubungan antara durasi buzzer dan periode sinyal tetap mengacu pada prinsip dasar frekuensi dalam fisika:

$$f = \frac{1}{T}$$

Zenius, (2021)

Semakin panjang nilai durasi yang disimpan, semakin rendah frekuensi bunyi buzzer yang dihasilkan.

- Kesimpulan Pengujian

Pengujian buzzer menggunakan EEPROM menunjukkan hasil sebagai berikut:

1. Sistem berhasil membaca dan menulis data durasi ke memori *non-volatile*.
2. Nilai durasi tetap tersimpan walaupun perangkat di *reset* atau dimatikan.

3. Pengguna dapat mengubah durasi dengan mudah melalui Serial Monitor tanpa perlu memprogram ulang mikrokontroler.
4. Fungsi `EEPROM.put()` dan `EEPROM.commit()` bekerja dengan stabil, memastikan data tersimpan permanen.

Dengan demikian, integrasi EEPROM pada sistem buzzer ini terbukti efektif untuk penyimpanan konfigurasi jangka panjang dalam aplikasi kontrol IoT berbasis ESP32.

C. Pengujian EEPROM untuk *setting* Buzzer *ON/OFF*

1) Deskripsi Program

Program ini dirancang untuk mengontrol pola bunyi buzzer menggunakan metode *non-blocking timer* (`millis()`) dan menyimpan konfigurasi durasi *ON* dan *OFF* ke dalam EEPROM agar pengaturan tetap tersimpan meskipun perangkat dimatikan. Dengan demikian, pengguna dapat mengubah pola bunyi buzzer melalui Serial Monitor dan sistem akan menyimpan pengaturan terakhir secara otomatis ke dalam memori permanen.

Fitur utama dari program ini meliputi:

1. Pengaturan pola bunyi buzzer berdasarkan tiga mode (‘a', ‘b', dan ‘c').
2. Penggunaan EEPROM untuk menyimpan durasi *ON/OFF* secara permanen.
3. Penggunaan millis() sebagai sistem waktu *non-blocking* agar program tidak terganggu oleh fungsi delay().
4. Komunikasi interaktif dengan pengguna melalui Serial Monitor.

2) Kodingan Program

```
#include <EEPROM.h>

#define BUZZER_PIN 15        // Pin buzzer (LOW trigger)
#define EEPROM_SIZE 512      // Ukuran EEPROM (wajib di-
EEPROM.begin())
#define ADDR_DURASI_ON 0     // Alamat EEPROM untuk durasi
ON
#define ADDR_DURASI_OFF 4    // Alamat EEPROM untuk durasi
OFF

unsigned long durasiOn;      // Durasi ON buzzer dalam ms
unsigned long durasiOff;      // Durasi OFF buzzer dalam
ms
unsigned long waktuSebelumnya = 0; // Untuk menyimpan
waktu terakhir
bool isBuzzerOn = false;     // Status buzzer (ON atau
OFF)

void setup() {
  Serial.begin(115200);
  pinMode(BUZZER_PIN, OUTPUT);
  // Pastikan buzzer mati saat startup (karena low
trigger)
  digitalWrite(BUZZER_PIN, HIGH);

  // Inisialisasi EEPROM
  if (!EEPROM.begin(EEPROM_SIZE)) {
    Serial.println("Gagal inisialisasi EEPROM!");
    while (1);
  }
```

```
  // Membaca data lama dari EEPROM
  EEPROM.get(ADDR_DURASI_ON, durasiOn);
  EEPROM.get(ADDR_DURASI_OFF, durasiOff);

  // Jika belum ada data, set nilai default ke pola 'a'
  if (durasiOn == 0xFFFFFFFF || durasiOn == 0) {
    durasiOn = 1000;
  }
  if (durasiOff == 0xFFFFFFFF || durasiOff == 0) {
    durasiOff = 1000;
  }

  // Menyimpan nilai default ke EEPROM jika belum ada
  EEPROM.put(ADDR_DURASI_ON, durasiOn);
  EEPROM.put(ADDR_DURASI_OFF, durasiOff);
  EEPROM.commit();

Serial.println("====================================");
  Serial.println("Konfigurasi Buzzer dari EEPROM:");
  Serial.print("Durasi ON: ");
  Serial.print(durasiOn);
  Serial.println(" ms");
  Serial.print("Durasi OFF: ");
  Serial.print(durasiOff);
  Serial.println(" ms");

Serial.println("====================================");
  Serial.println("Masukkan 'a', 'b', atau 'c' untuk
mengubah pola:");
  Serial.println("- 'a': ON 1000 ms, OFF 1000 ms");
```

```
  Serial.println("- 'b': ON 2000 ms, OFF 1000 ms");
  Serial.println("- 'c': ON 1000 ms, OFF 2000 ms");

Serial.println("====================================");
}

void loop() {
  unsigned long waktuSekarang = millis();

  // Kontrol buzzer menggunakan millis()
  if (isBuzzerOn && waktuSekarang - waktuSebelumnya >=
durasiOn) {
    digitalWrite(BUZZER_PIN, HIGH); // Buzzer OFF (LOW
trigger)
    isBuzzerOn = false;
    waktuSebelumnya = waktuSekarang;
  } else if (!isBuzzerOn && waktuSekarang -
waktuSebelumnya >= durasiOff) {
    digitalWrite(BUZZER_PIN, LOW);  // Buzzer ON (LOW
trigger)
    isBuzzerOn = true;
    waktuSebelumnya = waktuSekarang;
  }

  // Membaca input dari Serial Monitor
  if (Serial.available()) {
    char input = Serial.read();

    // Periksa input 'a', 'b', atau 'c'
    if (input == 'a' || input == 'b' || input == 'c') {
```

```
            // Atur durasi ON dan OFF berdasarkan pilihan
            if (input == 'a') {
              durasiOn = 1000;
              durasiOff = 1000;
            } else if (input == 'b') {
              durasiOn = 2000;
              durasiOff = 1000;
            } else if (input == 'c') {
              durasiOn = 1000;
              durasiOff = 2000;
            }

            // Simpan durasi baru ke EEPROM
            EEPROM.put(ADDR_DURASI_ON, durasiOn);
            EEPROM.put(ADDR_DURASI_OFF, durasiOff);
            EEPROM.commit();

            Serial.println("\n---------------------------------
        -----");
            Serial.print("Konfigurasi baru tersimpan: ");
            Serial.print("Pola '");
            Serial.print(input);
            Serial.println("'");
            Serial.print("Durasi ON: ");
            Serial.print(durasiOn);
            Serial.println(" ms");
            Serial.print("Durasi OFF: ");
            Serial.print(durasiOff);
            Serial.println(" ms");
            Serial.println("---------------------------------
        ---");
```

```
    } else {
      Serial.println("\nInput tidak valid. Masukkan 'a',
'b', atau 'c'.");
    }
  }
}
```

3) Penjelasan Program

Kode di atas menggunakan EEPROM sebagai penyimpanan nilai durasi *ON* dan *OFF* buzzer secara permanen, sehingga ketika sistem dimatikan dan dihidupkan kembali, konfigurasi terakhir tetap tersimpan.

A. Bagian Deklarasi dan Setup

i. `#include <EEPROM.h>` digunakan untuk mengakses fungsi baca/tulis memori EEPROM.

ii. Pin buzzer didefinisikan pada pin 15, dengan logika *LOW trigger* (aktif ketika *LOW*).

iii. EEPROM memiliki dua alamat memori: `ADDR_DURASI_ON` untuk menyimpan waktu *ON* dan `ADDR_DURASI_OFF` untuk waktu *OFF*.

iv. Nilai default (1000 ms) akan ditetapkan jika EEPROM belum pernah diisi sebelumnya.

B. Bagian Loop (Logika Utama)

i. Program menggunakan fungsi `millis()` untuk mengukur waktu berjalan tanpa menghentikan proses lain (non-blocking).

ii. Jika buzzer menyala melebihi durasi *ON*, maka akan dimatikan. Sebaliknya, jika buzzer mati melebihi durasi *OFF*, maka akan dinyalakan.

iii. Status *ON/OFF* buzzer disimpan dalam variabel `isBuzzerOn`.

C. *Input* dari Serial Monitor

i. Pengguna dapat memasukkan huruf 'a', 'b', atau 'c' untuk mengubah pola:

1. 'a': *ON* 1000 ms, *OFF* 1000 ms
2. 'b': *ON* 2000 ms, *OFF* 1000 ms
3. 'c': *ON* 1000 ms, *OFF* 2000 ms

ii. Setiap kali pola diubah, nilai baru disimpan ke EEPROM agar tetap tersimpan meski perangkat dimatikan.

4) Hasil dan Tampilan *Output*

Berikut tampilan hasil *output* pada Serial Monitor saat program dijalankan:

```
12:10:47.161 -> Konfigurasi Buzzer dari EEPROM:
12:10:47.161 -> Durasi ON: 1000 ms
12:10:47.161 -> Durasi OFF: 1000 ms
```

Gambar 4.5.5.5 *Output* Konfigurasi Buzzer Dari EEPROM

Jika ingin memasukkan ‘a','b','c' untuk mengubah pola maka bisa dilihat pada gambar di bawah ini:

```
12:10:47.161 -> Masukkan 'a', 'b', atau 'c' untuk mengubah pola:
12:10:47.161 -> - 'a': ON 1000 ms, OFF 1000 ms
12:10:47.161 -> - 'b': ON 2000 ms, OFF 1000 ms
12:10:47.161 -> - 'c': ON 1000 ms, OFF 2000 ms
```

Gambar 4.5.5.5 *Output* masukkan ‘a','b','c' untuk mengubah pola

Saat pengguna mengetik huruf a','b','c' ‘pada Serial Monitor, maka *output* berubah menjadi:

```
12:11:41.403 -> Konfigurasi baru tersimpan: Pola 'a'
12:11:41.403 -> Durasi ON: 1000 ms
12:11:41.403 -> Durasi OFF: 1000 ms
```

Gambar 4.5.5.5 *Output* Konfigurasi baru tersimpan pada Pola ‘a'

```
12:13:44.778 -> Konfigurasi baru tersimpan: Pola 'b'
12:13:44.778 -> Durasi ON: 2000 ms
12:13:44.778 -> Durasi OFF: 1000 ms
```

Gambar 4.5.5.5 *Output* Konfigurasi baru tersimpan pada Pola ‘b'

```
12:14:45.791 -> Konfigurasi baru tersimpan: Pola 'c'
12:14:45.791 -> Durasi ON: 1000 ms
12:14:45.791 -> Durasi OFF: 2000 ms
```

Gambar 4.5.5.5 *Output* Konfigurasi baru tersimpan pada Pola ‘c'

Buzzer kemudian berbunyi mengikuti pola tersebut, dan konfigurasi tersimpan otomatis di EEPROM.

5) Kesimpulan

Dari implementasi program ini dapat disimpulkan bahwa:

a. EEPROM berfungsi dengan baik sebagai penyimpan data permanen pada ESP32.
b. Penggunaan `millis()` berhasil membuat sistem bekerja tanpa *delay blocking*.
c. Pengguna dapat dengan mudah mengubah pola kerja buzzer melalui Serial Monitor, dan sistem menyimpannya secara otomatis.
d. Program ini efisien, responsif, dan cocok diterapkan untuk aplikasi peringatan suara otomatis atau sistem notifikasi berbasis IoT.

4.5.11 Pengujian Buzzer Menggunakan Koneksi WIFI

A. Tujuan Pengujian

Tujuan dari pengujian ini adalah untuk memastikan bahwa buzzer dapat berfungsi secara *non-blocking* (tanpa delay) saat memantau koneksi WiFi pada modul ESP32. Dengan cara ini, buzzer tetap dapat berfungsi selama proses pengecekan atau penyambungan ulang WiFi.

B. Alat dan Bahan

1. ESP32 DevKit V1
2. Buzzer aktif (*Low Trigger*)
3. Kabel *jumper*
4. Jaringan WiFi HOSTPOT melalui HP saya dengan SSID "Alan" dan password "alan12378"
5. *Software* Arduino IDE

C. Langkah Pengujian

1. Hubungkan pin buzzer ke GPIO 15 ESP32.
2. Upload program berikut ke ESP32:

```
#include <WiFi.h>

#define BUZZER_PIN 15     // pin buzzer (low
trigger)

const char* ssid = "Alan";
const char* password = "alan12378";

unsigned long buzzerOn = 1000;
unsigned long buzzerOff = 1000;
unsigned long prevBuzzerMillis = 0;
bool buzzerState = false;

unsigned long prevWiFiMillis = 0;
```

```
unsigned long intervalWiFi = 5000;

void setup() {
  Serial.begin(115200);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH); // buzzer OFF (low
trigger)
}

void loop() {
  unsigned long currentMillis = millis();

  // ====== BUZZER Non-blocking ======
  if     (!buzzerState     &&     currentMillis     -
prevBuzzerMillis >= buzzerOff) {
    buzzerState = true;
    prevBuzzerMillis = currentMillis;
    digitalWrite(BUZZER_PIN, LOW);  // buzzer ON
  }
  else    if    (buzzerState    &&    currentMillis    -
prevBuzzerMillis >= buzzerOn) {
    buzzerState = false;
    prevBuzzerMillis = currentMillis;
    digitalWrite(BUZZER_PIN, HIGH); // buzzer OFF
  }

  // ====== WiFi Reconnect Non-blocking ======
  if (currentMillis - prevWiFiMillis >= intervalWiFi)
{
    prevWiFiMillis = currentMillis;
    if (WiFi.status() != WL_CONNECTED) {
```

```
    Serial.println("WiFi terputus, mencoba
koneksi...");
        WiFi.disconnect();
    WiFi.begin(ssid, password);
       }
   else {
      Serial.println("WiFi masih terhubung Д");
       }
  }
 }
```

3. Buka Serial Monitor untuk memantau status koneksi WiFi.
4. Amati bunyi buzzer yang menyala dan mati secara bergantian setiap 1 detik.
5. Putuskan koneksi WiFi untuk menguji fungsi *reconnect* otomatis.

Hasil *output*nya Bisa dilihat di bawah ini:

```
12:05:04.855 -> mode:DIO, clock div:1
12:05:04.898 -> load:0x3fff0030,len:4744
12:05:04.898 -> load:0x40078000,len:15672
12:05:04.898 -> load:0x40080400,len:3152
12:05:04.898 -> entry 0x4008059c
12:05:10.160 -> WiFi terputus, mencoba koneksi...
12:05:15.188 -> WiFi masih terhubung ✅
```

Gambar 4.5.5.6 Hasil *output* Koneksi WIFI

5. Kesimpulan

Berdasarkan hasil pengujian, sistem buzzer bekerja secara independen dari koneksi WiFi. Fungsi non-blocking menggunakan `millis()` terbukti efektif, karena ESP32 dapat melakukan dua proses paralel:

i. Mengontrol buzzer menyala dan mati secara teratur.
ii. Memeriksa serta menyambungkan ulang WiFi setiap 5 detik.

Dengan demikian, sistem ini stabil dan efisien untuk digunakan dalam proyek IoT berbasis WiFi.

4.5.12 Pengujian Buzzer Menggunakan Koneksi MQTT

A. Tujuan Pengujian

Tujuan dari pengujian ini adalah untuk memastikan bahwa sistem dapat melakukan pengendalian buzzer secara non-blocking dan menjaga koneksi WiFi dan MQTT secara otomatis. Selain itu, pengujian ini memastikan bahwa broker publik HiveMQ memiliki kemampuan untuk mengendalikan buzzer dari jarak jauh melalui protokol MQTT.

B. Alat dan Bahan

i. ESP32 DevKit V1
ii. Buzzer aktif (*low trigger*)
iii. kabel *jumper* (*female to female*)
iv. Koneksi internet WiFi di rumah saya (SSID: samudra43, Password: TNIAL.25.)
v. *Software* Arduino IDE
vi. MQTT *Dashboard* (IoT MQTT Panel di Android)

A. Langkah Pengujian

i. Hubungkan buzzer aktif ke pin GPIO 15 pada ESP32.
ii. Upload program berikut ke board ESP32:

```
#include <WiFi.h>
#include <PubSubClient.h>

// --- Konfigurasi WiFi ---
const char* ssid = "samudra43";
const char* password = "TNIAL.25.";

// --- Konfigurasi MQTT (HiveMQ Public Broker) ---
     const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
const char* topic_sub = "esp32/buzzer"; // Topic subscribe
const char* topic_pub = "esp32/status"; // Topic publish

#define BUZZER_PIN 15   // Pin buzzer (low trigger)

WiFiClient espClient;
PubSubClient client(espClient);

// --- Variabel kontrol buzzer ---
bool buzzerActive = false;
```

```
unsigned long previousMillis = 0;
const unsigned long buzzerOnTime = 500;   // 0.5 detik ON
const unsigned long buzzerOffTime = 9500; // 9.5 detik OFF
bool buzzerState = false; // LOW = aktif, HIGH = mati

// --- Variabel pengecekan koneksi ---
unsigned long lastCheckMillis = 0;
const unsigned long checkInterval = 30000; // 30 detik

// ---------------- MQTT CALLBACK ----------------
 void callback(char* topic, byte* message, unsigned int length) {
  String msg;
  for (int i = 0; i < length; i++) {
    msg += (char)message[i];
  }
  Serial.print("Pesan diterima [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(msg);

  if (msg == "ON") {
    buzzerActive = true;
    buzzerState = false; // mulai dari buzzer mati
    previousMillis = millis();
    client.publish(topic_pub, "Buzzer ON");
  } else if (msg == "OFF") {
    buzzerActive = false;
    digitalWrite(BUZZER_PIN, HIGH); // pastikan buzzer mati
    client.publish(topic_pub, "Buzzer OFF");
  }
}

// ---------------- WIFI CONNECT ----------------
void connectWiFi() {
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Menghubungkan ke WiFi ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
  }
}

// ---------------- MQTT RECONNECT ----------------
void connectMQTT() {
  if (!client.connected() && WiFi.status() == WL_CONNECTED) {
    Serial.print("Menghubungkan ke MQTT...");
    if (client.connect("ESP32_BuzzerClient")) {
      Serial.println("connected");
```

```
      client.subscribe(topic_sub);
      client.publish(topic_pub, "ESP32 Connected");
    } else {
      Serial.print("Gagal, rc=");
      Serial.println(client.state());
    }
  }
}

// ---------------- SETUP ----------------
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH); // buzzer off (low trigger)

  Serial.begin(115200);
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
}

// ---------------- LOOP ----------------
void loop() {
  unsigned long currentMillis = millis();

  // --- Cek koneksi WiFi & MQTT tiap 30 detik ---
  if (currentMillis - lastCheckMillis >= checkInterval) {
    lastCheckMillis = currentMillis;
     connectWiFi();
    connectMQTT();
  }

  // --- Jalankan MQTT hanya jika sudah connect ---
  if (client.connected()) {
    client.loop();
  }

  // --- Logika buzzer ---
  if (buzzerActive) {
    if (!buzzerState && (currentMillis - previousMillis >=
buzzerOffTime)) {
      digitalWrite(BUZZER_PIN, LOW);
      buzzerState = true;
      previousMillis = currentMillis;
      Serial.println("Buzzer ON");

      if (client.connected()) {
        client.publish(topic_pub, "Buzzer ON");
      }
```

```
    }
    else if (buzzerState && (currentMillis - previousMillis >=
buzzerOnTime)) {
      digitalWrite(BUZZER_PIN, HIGH);
      buzzerState = false;
      previousMillis = currentMillis;
      Serial.println("Buzzer OFF");

      if (client.connected()) {
        client.publish(topic_pub, "Buzzer OFF");
      }
    }
  }
}
```

iii. Setelah program berhasil diunggah, buka Serial Monitor pada baud rate 115200.

iv. Tunggu hingga muncul pesan di Arduino IDE :

`Menghubungkan ke WiFi samudra43`

`Menghubungkan ke MQTT... connected`

`ESP32 Connected`

yang menandakan ESP32 berhasil terhubung ke jaringan WiFi dan broker MQTT.

v. Buka aplikasi IoT MQTT Panel di Android, kemudian tambahkan *dashboard* baru dengan pengaturan berikut:

1. *Broker* : broker.hivemq.com
2. *Port* : 1883
3. *Subscribe Topic* : esp32/status
4. *Publish Topic* : `esp32/buzzer`
5. *Widget* :

a) *Toggle Button* untuk mengirim perintah *ON/OFF* ke *topic publish* (`esp32/buzzer`).

b) *Text Log* untuk menampilkan pesan yang diterima dari broker, khususnya dari *topic* yang kamu *subscribe* (`esp32/status`).

D. Hasil Pengujian

1) Tampilan hasil di Serial Monitor:

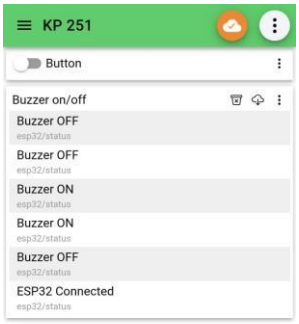
Gambar 4.5.5.7 *Output* Koneksi MQTT di Arduino IDE

2) Tampilan hasil di aplikasi IoT MQTT Panel:

A. Saat Posisi Buzzer *ON*

Gambar 4.5.5.7 *Output* Koneksi MQTT di aplikasi IoT MQTT Panel Saat Posisi Buzzer *ON*

B. Saat Posisi Buzzer *OFF*



Gambar 4.5.5.7 *Output* Koneksi MQTT di aplikasi IoT MQTT Panel Saat Posisi Buzzer *OFF*

E. Kesimpulan

Berdasarkan hasil pengujian sistem kendali buzzer berbasis ESP32 dan MQTT, dapat disimpulkan bahwa:

1. Sistem berhasil terhubung secara stabil ke jaringan WiFi dan broker MQTT (broker.hivemq.com) menggunakan port 1883, serta mampu menjaga koneksi secara otomatis saat terjadi pemutusan.
2. Komunikasi antara ESP32 dan aplikasi IoT MQTT Panel berjalan dengan baik melalui topik:

A. *Publish Topic:* esp32/buzzer untuk mengirim perintah dari pengguna.

B. *Subscribe Topic:* esp32/status untuk menerima umpan balik kondisi buzzer.

3. Fitur *non-blocking buzzer control* berfungsi dengan benar — buzzer dapat menyala dan mati berkala tanpa mengganggu proses koneksi WiFi maupun MQTT.
4. Dari hasil uji di Serial Monitor dan IoT MQTT Panel, terlihat bahwa setiap perintah *ON/OFF* yang dikirim dari *dashboard* diterima dan dieksekusi sesuai dengan respon yang ditampilkan.

5. Dengan demikian, sistem ini telah berhasil menjalankan fungsi pengendalian buzzer jarak jauh secara *real-time* menggunakan protokol MQTT, dan siap dikembangkan untuk kontrol perangkat IoT lainnya.

4.6 Perancangan Rangkaian

4.6.1 Skenario Pengujian Rangkaian

Pengujian dilakukan untuk memastikan bahwa sistem pemberian pakan otomatis dan notifikasi IoT bekerja dengan benar. Skenario pengujian adalah sebagai berikut:

A. Server MQTT mensimulasikan jadwal pemberian pakan
B. Broker MQTT mengirimkan pesan ke mikrokontroler ESP32 ketika waktu jadwal tercapai.
C. ESP32 kemudian memicu buzzer agar berbunyi sebagai bentuk notifikasi.
D. Buzzer berfungsi sebagai alarm simulasi, sehingga dosen maupun mahasiswa dapat mengetahui waktu pemberian pakan tanpa perlu memantau akuarium fisik secara langsung.

4.6.2 Implementasi Simulasi di Ruang Dosen

Beberapa komponen utama rangkaian sistem digunakan sebagai proyek demonstrasi, seperti:

1. ESP32 DevKit V1
2. Buzzer dengan tipe *trigger low-level*
3. Koneksi WiFi dan broker MQTT sebagai media komunikasi data

Pada tahap ini, akuarium fisik tidak digunakan. Sebaliknya, simulasi penuh dilakukan melalui sistem IoT, dengan detail berikut:

1. Jadwal pemberian pakan disimulasikan melalui server MQTT
2. Status buzzer dapat dipantau melalui aplikasi *smartphone* atau MQTT *Dashboard*, seperti HiveMQ *Dashboard*.
3. *Dashboard* tersebut memungkinkan dosen dan mahasiswa melihat status sistem secara *real-time*, yang menunjukkan bahwa logika kontrol dan komunikasi antarperangkat berjalan dengan baik.

4.7 Pengujian Rangkaian *Project* Akhir

4.7.1 Deskripsi Umum Pengujian

Pengujian rangkaian dilakukan untuk memastikan bahwa perangkat keras dan perangkat lunak yang telah dibuat sesuai dengan sistem pemberian pakan otomatis berbasis IoT. Dua jenis perangkat utama digunakan dalam *project* akhir, yaitu:

1. Perangkat Keras (*Hardware*):

A. Modul mikrokontroler ESP32 DevKit V1

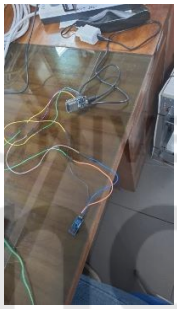
B. Buzzer tipe *low-level trigger*

C. Kabel *jumper male to male* dan *female to female*

D. Kabel *charger* mikrokontroler

E. Adaptor sebagai sumber daya

Seluruh perangkat dirangkai menjadi satu kesatuan sistem seperti pada gambar rancangan rangkaian.



Gambar 4.7.1 Rangkaian Project Akhir Perangkat Keras (*Hardware*)

2. Perangkat Lunak (*Software*)

Dengan menggunakan Arduino IDE di laptop, dapat menulis kode dan mengirimkannya ke mikrokontroler ESP32. Kemudian *Smartphone* menggunakan aplikasi IoT MQTT Panel untuk melacak status sistem dan mengendalikannya secara *real-time*.

4.7.2 Skenario Pengujian

Pengujian dilakukan untuk mengamati respon sistem terhadap pesan MQTT yang dikirim dari server dan memastikan bahwa proses notifikasi bekerja dengan baik. Tahapan skenario pengujian adalah sebagai berikut:

A. Jadwal pemberian pakan disimulasikan melalui server MQTT.

B. Ketika waktu jadwal tercapai, broker MQTT mengirimkan pesan ke ESP32.
C. ESP32 memicu buzzer untuk berbunyi sebagai tanda notifikasi waktu pemberian pakan.
D. Buzzer berfungsi sebagai alarm simulasi, yang membantu dosen dan mahasiswa mengetahui waktu pemberian pakan tanpa harus memantau akuarium fisik.

4.7.3 Implementasi Pengujian

Rangkaian sistem diuji di ruang dosen dengan konfigurasi sebagai berikut:

a) ESP32 terhubung ke jaringan WiFi UNDIKANet dengan kredensial yang telah diatur.
b) Broker MQTT yang digunakan adalah mqtt.dinamika.ac.id dengan port 1883.
c) Topik komunikasi yang digunakan mencakup:

i. `mhs/jadwal` → menerima perubahan jadwal dari server
ii. `mhs/waktu` → mengirim status waktu mundur ke *dashboard*
iii. `mhs/status` → mengirim status sistem (Aktif, *Idle*, atau Jadwal Diubah)

Pengujian dilakukan tanpa menggunakan akuarium fisik, melainkan hanya melalui simulasi logika dan kontrol dengan memanfaatkan buzzer sebagai indikator notifikasi serta status sistem yang muncul pada MQTT *Dashboard*. Dosen dan mahasiswa dapat memantau kondisi sistem secara *real-time* melalui *dashboard*, membuktikan bahwa komunikasi dua arah antara server MQTT dan ESP32 berjalan dengan baik.

4.7.4 Kodingan ARDUINO IDE

```
#include <PubSubClient.h>
#include <EEPROM.h>

// ------------------- Konfigurasi EEPROM ----------------
---
#define EEPROM_SIZE     16
#define ADDR_BANYAK     0
#define ADDR_LAMAMS     4
#define ADDR_BANYAK_DEF 8
#define ADDR_LAMAMS_DEF 12
```

```
// ------------------- Pin Buzzer -------------------
   #define BUZZER_PIN 15  // sesuaikan pin buzzer

// ------------------- Konfigurasi WiFi ------------------
-
const char* WIFI_SSID     = "UNDIKANet";
const char* WIFI_PASSWORD = "SemangatPagi:)";

// ------------------- Konfigurasi MQTT ------------------
-
const char* MQTT_SERVER   = "mqtt.dinamika.ac.id";
const int   MQTT_PORT     = 1883;
const char* MQTT_USER     = "mhs";
const char* MQTT_PASS     = "mahasiswa";

// ------------------- MQTT Topics -------------------
String baseTopic    = String(MQTT_USER) + "/";
String topicSub     = baseTopic + "jadwal";
String topicWaktu   = baseTopic + "waktu";
String topicStatus  = baseTopic + "status";

WiFiClient espClient;
PubSubClient client(espClient);

// ------------------- Variabel Jadwal -------------------
int Banyak;
int LamaMS;
unsigned long JadwalJamMS;
unsigned long prevMillis = 0;
unsigned long countdownMS = 0;

// ------------------- Variabel Buzzer -------------------
bool buzzerAktif = false;
unsigned long buzzerStart = 0;

// ------------------- Variabel tambahan ------------------
    -
bool wifiTerhubung = false;
unsigned long prevWiFiPrint = 0;

// ------------------- Fungsi EEPROM -------------------
void saveEEPROM(int addr, int value) {
  EEPROM.put(addr, value);
  EEPROM.commit();
}

int readEEPROM(int addr) {
```

```
  int val;
  EEPROM.get(addr, val);
  return val;
}

void loadJadwal() {
  Banyak = readEEPROM(ADDR_BANYAK);
  LamaMS = readEEPROM(ADDR_LAMAMS);
  if (Banyak <= 0) Banyak = 1;
  JadwalJamMS = (unsigned long)(24.0 * 3600 * 1000 /
    Banyak);
  countdownMS = JadwalJamMS;
}

// ------------------- MQTT Callback ------------------
void callback(char* topic, byte* payload, unsigned int
    length) {
  String msg;
  for (unsigned int i = 0; i < length; i++) msg +=
    (char)payload[i];
  msg.trim();

  if (msg.startsWith("UBAH:")) {
    int comma = msg.indexOf(',');
    if (comma > 0) {
      Banyak = msg.substring(5, comma).toInt();
      LamaMS = msg.substring(comma + 1).toInt();
      saveEEPROM(ADDR_BANYAK, Banyak);
      saveEEPROM(ADDR_LAMAMS, LamaMS);
      loadJadwal();
      if(client.connected())
    client.publish(topicStatus.c_str(), "Jadwal Diubah");
    }
  } else if (msg.equalsIgnoreCase("RESET")) {
    Banyak = readEEPROM(ADDR_BANYAK_DEF);
    LamaMS = readEEPROM(ADDR_LAMAMS_DEF);
    saveEEPROM(ADDR_BANYAK, Banyak);
    saveEEPROM(ADDR_LAMAMS, LamaMS);
    loadJadwal();
    if(client.connected())
    client.publish(topicStatus.c_str(), "Jadwal Reset");
  }
}

// ------------------- Setup WiFi & MQTT ----------------
    -
void reconnectWiFi() {
```

```
  if (WiFi.status() != WL_CONNECTED) {
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  }
}

void reconnectMQTT() {
  if (!client.connected() && wifiTerhubung) {
    String clientId = "ESP32Client_" +
    String(WiFi.macAddress());
     if (client.connect(clientId.c_str(), MQTT_USER,
    MQTT_PASS)) {
       Serial.println("MQTT Reconnected!");
       client.subscribe(topicSub.c_str());
     } else {
       Serial.print("MQTT Gagal, rc=");
       Serial.println(client.state());
     }
  }
}

// ------------------- Setup -------------------
void setup() {
  Serial.begin(115200);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, HIGH); // default buzzer OFF

  EEPROM.begin(EEPROM_SIZE);

  if (readEEPROM(ADDR_BANYAK_DEF) == 0) {
    saveEEPROM(ADDR_BANYAK_DEF, 2);
    saveEEPROM(ADDR_LAMAMS_DEF, 500);
    saveEEPROM(ADDR_BANYAK, 2);
    saveEEPROM(ADDR_LAMAMS, 500);
  }

  loadJadwal();

  // Mulai koneksi WiFi (non-blocking)
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.println("Menghubungkan ke WiFi...");
}

// ------------------- Loop -------------------
void loop() {
  static unsigned long lastWiFiCheck = 0;
  static unsigned long lastMQTTCheck = 0;
  unsigned long now = millis();
```

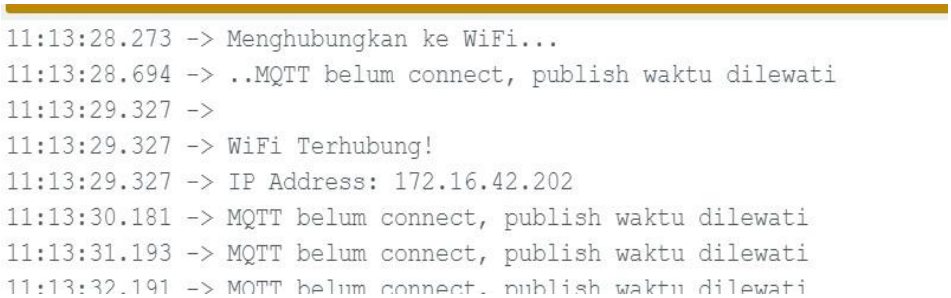
```
// ------------------- WiFi non-blocking ----------------
  ---
if (!wifiTerhubung) {
  if (WiFi.status() == WL_CONNECTED) {
    wifiTerhubung = true;
    Serial.println("\nWiFi Terhubung!");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    // Setup MQTT setelah WiFi connect
    client.setServer(MQTT_SERVER, MQTT_PORT);
    client.setCallback(callback);
  } else if (now - prevWiFiPrint >= 500) {
    Serial.print(".");
    prevWiFiPrint = now;
  }
}

// ------------------- Reconnect WiFi & MQTT ------------
  -------
if (now - lastWiFiCheck > 30000) {
  lastWiFiCheck = now;
  reconnectWiFi();
}

if (now - lastMQTTCheck > 10000) {
  lastMQTTCheck = now;
  reconnectMQTT();
}

// ------------------- MQTT Loop --------------------
if(client.connected()) client.loop();

// ------------------- Hitung mundur & logika buzzer ----
  ---------------
if (now - prevMillis >= 1000) {
  prevMillis = now;
  if (countdownMS >= 1000) countdownMS -= 1000;
  else countdownMS = 0;

  int jam   = (countdownMS / 3600000);
  int menit = (countdownMS % 3600000) / 60000;
  int detik = (countdownMS % 60000) / 1000;

  char buffer[32];
  sprintf(buffer, "%02d:%02d:%02d", jam, menit, detik);
```

```
    if(client.connected()) {
      client.publish(topicWaktu.c_str(), buffer);
    } else {
      Serial.println("MQTT belum connect, publish waktu
   dilewati");
    }
  }

  if (countdownMS == 0 && !buzzerAktif) {
    buzzerAktif = true;
    buzzerStart = now;
    digitalWrite(BUZZER_PIN, LOW);
    if(client.connected())
   client.publish(topicStatus.c_str(), "Buzzer Aktif");
   else Serial.println("MQTT belum connect, publish Buzzer
   Aktif dilewati");
  }

  if (buzzerAktif && (now - buzzerStart >= (unsigned
    long)LamaMS)) {
    digitalWrite(BUZZER_PIN, HIGH);
    buzzerAktif = false;
    countdownMS = JadwalJamMS;
    if(client.connected())
   client.publish(topicStatus.c_str(), "Idle");
    else Serial.println("MQTT belum connect, publish Idle
   dilewati");
  }
}
```

4.7.5 *Output* dan Pembahasan

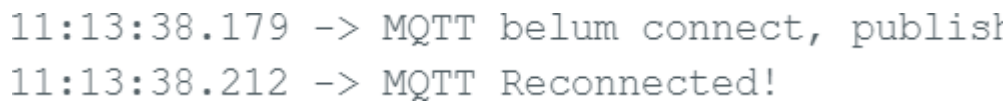
a. Hasil Pengujian Koneksi WiFi dan MQTT

Hasil pengujian awal pada Serial Monitor menunjukkan bahwa sistem berhasil melakukan koneksi ke jaringan WiFi dan memperoleh alamat IP (172.16.42.202). Namun, sebelum koneksi MQTT stabil, sistem sempat menampilkan pesan:



Gambar 4.7.5 *Output* Ketika sebelum koneksi MQTT stabil di ARDUINO IDE

Setelah beberapa kali percobaan, sistem berhasil melakukan *reconnect* MQTT seperti terlihat pada *log*:



Gambar 4.7.5 *Output* MQTT *reconnect* di ARDUINO IDE

Hal ini membuktikan bahwa mekanisme *reconnect* otomatis pada kode program bekerja dengan baik ketika koneksi ke broker MQTT sempat terputus.

b. Hasil Pengujian melalui Aplikasi IoT MQTT Panel

Setelah koneksi berhasil, sistem diuji menggunakan aplikasi IoT MQTT Panel. Terdapat tiga topik komunikasi utama yang digunakan:

1. mhs/jadwal → menerima perintah dari pengguna (UBAH / *RESET*).
2. mhs/status → menampilkan status buzzer (“Idle”, “Buzzer Aktif”, “Jadwal Diubah”, “Jadwal *Reset*”).
3. mhs/waktu → menampilkan waktu hitung mundur (countdown) menuju jadwal bunyi berikutnya.

Tampilan pada panel menunjukkan perubahan status secara *real-time* sesuai logika program:

a) Saat countdown berjalan, status menampilkan “*Idle*”.

Gambar 4.7.5 Saat Posisi Buzzer *Idle*

b) Saat waktu habis (00:00:00), status berubah menjadi "Buzzer Aktif" selama 0,5 detik.

Gambar 4.7.5 Status Buzzer Aktif

c) Setelah buzzer selesai aktif, sistem kembali ke "*Idle*" dan memulai perhitungan ulang.

d) Jika perintah *RESET* dikirim, status berubah menjadi "Jadwal *Reset*" dan *countdown* diatur ulang.

Gambar 4.7.5 Jadwal *Reset*

e) Jika perintah UBAH dikirim, status menjadi **"**Jadwal Diubah**"** dan nilai jadwal baru tersimpan di EEPROM.

Gambar 4.7.5 Ubah Jadwal

c. Analisis Perhitungan Jadwal dan Interval

Rumus dasar perhitungan interval antar bunyi buzzer mengacu pada konsep pembagian waktu dalam satu hari terhadap jumlah kejadian yang diinginkan. Secara matematis dapat dinyatakan sebagai berikut:

$$\text{JadwalJamMS} = \frac{24 \times 3600 \times 1000}{\text{Banyak}}$$

Dengan keterangan:

- **Banyak** = jumlah bunyi buzzer dalam satu hari
- **LamaMS** = durasi buzzer aktif dalam milidetik (ms)
- 1 hari = 86.400.000 ms

Konsep periode dan frekuensi pada gelombang mirip (Zenius, 2021). Pada gelombang, periode adalah kebalikan dari frekuensi, yaitu waktu yang dibutuhkan untuk satu siklus kejadian. Dalam hal ini, setiap "siklus" adalah satu kali buzzer menyala setiap hari. Oleh karena itu, semakin besar jumlah "siklus", semakin pendek waktu yang dihabiskan untuk menunggu bunyi buzzer.

Sebagai Contoh :

Jika perintah yang dikirim melalui aplikasi adalah:

`UBAH:2150,500`

maka:

$$\frac{86.400.000}{2150} = 40.186\,ms \approx 40\,detik$$

Artinya, buzzer akan aktif setiap 40 detik selama 0,5 detik, kemudian kembali ke kondisi *Idle* hingga interval berikutnya tercapai.

d. Tabel Hasil Uji Jadwal dan Status Buzzer

Tabel berikut menyajikan hasil uji coba perubahan jadwal, perhitungan waktu mundur (*countdown*), status buzzer, serta keterangan sistem dengan durasi bunyi buzzer sebesar 500 ms (0,5 detik). Perhitungan dilakukan berdasarkan rumus interval antar bunyi, proses *countdown*, konversi waktu ke format `jam:menit:detik`, reset waktu mundur, serta total durasi buzzer aktif (*ON*) selama pengujian.

Tabel 4.7.5 Hasil Uji Jadwal dan Status Buzzer

NO	Ubah Jadwal (Banyak, LamaMS)	Interval	Hitung Mundur di MQTT	Status Buzzer	Penjelasan
1	(50, 500)	28 menit 48 detik	00:28:48 → 00:00:00	Buzzer Aktif	Karena 24 jam dibagi 50 = 1728 detik (28 menit 48 detik). Jadi setiap 28:48 buzzer *ON* selama 0,5 detik.
2	(100, 500)	14 menit 24 detik	00:14:24 → 00:00:00	Buzzer Aktif	24 jam ÷ 100 = 864 detik (14:24). Jadi buzzer nyala tiap 14 menit sekali.
3	(150, 500)	9 menit 36 detik	00:09:36 → 00:00:00	Buzzer Aktif	24 jam ÷ 150 = 576 detik (9:36). Semakin rapat.
4	(200, 500)	7 menit 12 detik	00:07:12 → 00:00:00	Buzzer Aktif	24 jam ÷ 200 = 432 detik (7:12).
5	(250, 500)	5 menit 45 detik	00:05:45 → 00:00:00	Buzzer Aktif	24 jam ÷ 250 = 345,6 detik ≈ 5:45.
6	(300, 500)	4 menit 48 detik	00:04:48 → 00:00:00	Buzzer Aktif	24 jam ÷ 300 = 288 detik (4:48).
7	(350, 500)	4 menit 08 detik	00:04:08 → 00:00:00	Buzzer Aktif	24 jam ÷ 350 = 246,9 detik ≈ 4:08.
8	(400, 500)	3 menit 36 detik	00:03:36 → 00:00:00	Buzzer Aktif	24 jam ÷ 400 = 216 detik (3:36).

9	(450, 500)	3 menit 12 detik	00:03:12 → 00:00:00	Buzzer Aktif	24 jam ÷ 450 = 192 detik (3:12).
10	(500, 500)	2 menit 52 detik	00:02:52 → 00:00:00	Buzzer Aktif	24 jam ÷ 500 = 172,8 detik ≈ 2:52.
11	(1000, 500)	1 menit 26 detik	00:01:26 → 00:00:00	Buzzer Aktif	Semakin sering, tiap 86,4 detik buzzer ON.
12	(1500, 500)	57 detik	00:00:57 → 00:00:00	Buzzer Aktif	Nyala tiap 57 detik.
13	(2000, 500)	43 detik	00:00:43 → 00:00:00	Buzzer Aktif	Nyala tiap 43 detik.
14	(2500, 500)	34 detik	00:00:34 → 00:00:00	Buzzer Aktif	Nyala tiap 34 detik.
15	(3000, 500)	28 detik	00:00:28 → 00:00:00	Buzzer Aktif	Nyala tiap 28 detik.
16	(3500, 500)	24 detik	00:00:24 → 00:00:00	Buzzer Aktif	Nyala tiap 24 detik.
17	(4000, 500)	21 detik	00:00:21 → 00:00:00	Buzzer Aktif	Nyala tiap 21 detik.
18	(4500, 500)	19 detik	00:00:19 → 00:00:00	Buzzer Aktif	Nyala tiap 19 detik.
19	(5000, 500)	17 detik	00:00:17 → 00:00:00	Buzzer Aktif	Nyala tiap 17 detik.
20	(5500, 500)	16 detik	00:00:16 → 00:00:00	Buzzer Aktif	Nyala tiap 16 detik.

Analisis:

Jumlah banyak bunyi buzzer yang dibuat setiap hari, atau jumlah bunyi buzzer, terkait dengan waktu antara bunyi semakin pendek, seperti yang ditunjukkan dalam tabel di atas. Hal ini menunjukkan bahwa, berdasarkan perhitungan matematis rumus interval, sistem logika program berhasil menyesuaikan waktu antara bunyi. Selain itu, fungsi `millis()` dan mekanisme EEPROM bekerja dengan benar, dan *dashboard* MQTT menampilkan hasil hitung mundur yang konsisten. Saat waktu mundur mencapai 00:00:00, status "Buzzer Aktif" muncul.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian sistem Monitoring dan Notifikasi Jadwal Pemberian Pakan Ikan Akuarium berbasis ESP32 dan Buzzer dengan komunikasi MQTT, maka dapat diambil beberapa kesimpulan sebagai berikut:

1. **S**istem Monitoring dan Notifikasi Jadwal Pemberian Pakan Ikan Akuarium berbasis ESP32 dan Buzzer dengan komunikasi MQTT berhasil dirancang dan diimplementasikan di Ruang Dosen S1 Teknik Komputer Universitas Dinamika**.**
2. ESP32 berfungsi sebagai mikrokontroler utama yang mengatur pengiriman dan penerimaan data melalui protokol MQTT. Ini memungkinkan platform pemantauan untuk menerima notifikasi jadwal pakan secara *real-time*.
3. Buzzer bekerja sebagai indikator bunyi otomatis yang memberikan peringatan sesuai jadwal pemberian pakan yang telah ditentukan, berdasarkan hasil perhitungan interval waktu dan *countdown* yang telah diatur pada sistem.
4. Dengan menggunakan rumus konversi waktu harian (dalam milidetik), sistem dapat menghitung jadwal dan interval bunyi dengan akurat. Selain itu, sistem dapat menampilkan status buzzer aktif dan sisa waktu hitung mundur secara sinkron dengan data MQTT.
5. Hasil uji coba menunjukkan bahwa sistem dapat memberikan notifikasi tepat waktu dengan buzzer aktif selama 500 milidetik, yang menunjukkan bahwa desain sistem telah berjalan sesuai fungsi dan mendukung otomatisasi proses pemeliharaan ikan hias di akuarium kampus.

5.2. Saran

Berdasarkan hasil perancangan dan implementasi sistem Monitoring Notifikasi Jadwal Pemberian Pakan Ikan Akuarium Menggunakan ESP32 dan Buzzer Berbasis MQTT di Ruang Dosen S1 Teknik Komputer Universitas Dinamika, beberapa saran yang dapat diberikan untuk pengembangan selanjutnya adalah sebagai berikut:

1. Penambahan sensor otomatis seperti *feeding sensor* atau *ultrasonic sensor* dapat diterapkan agar sistem tidak hanya memberikan notifikasi, tetapi juga mampu mendeteksi kondisi pakan dan air secara *real-time*.
2. Mengintegrasikan dengan aktuator pemberi pakan otomatis, yang memungkinkan sistem untuk memberikan pakan secara otomatis sesuai jadwal yang telah ditentukan tanpa perlu melakukan perubahan manual.
3. Membangun antarmuka pengguna (UI/UX) untuk platform MQTT atau dashboard berbasis web atau mobile untuk membuat tampilan monitoring lebih interaktif dan menarik.
4. Meningkatkan kemampuan penyimpanan data, juga dikenal sebagai data logging, dengan menggunakan database atau layanan cloud untuk menyimpan sejarah jadwal pakan dan aktivitas buzzer untuk analisis jangka panjang.
5. Optimalkan konsumsi daya dan kestabilan koneksi Wi-Fi pada ESP32 untuk membuat sistem lebih efisien dan handal, terutama jika digunakan dalam jangka waktu yang lama.
6. Lakukan uji coba di berbagai kondisi jaringan dan suhu untuk mengetahui kinerja sistem di laboratorium atau di luar ruangan.

DAFTAR PUSTAKA

Arduino. (2023). *EEPROM Library Documentation*. Diakses dari: https://www.arduino.cc/en/Reference/EEPROM

AsyncMqttClient Library. (2023). GitHub Repository. Diakses dari: https://github.com/marvinroger/async-mqtt-client

Atmel. (2016). *ATmega328/P Datasheet*. Atmel Corporation.

Banks, A., & Gupta, R. (2014). *MQTT Version 3.1.1*. OASIS Standard.

Banzi, M., & Shiloh, M. (2014). *Getting Started with Arduino*. Maker Media.

Burhani, F., Zaenurrohman, Z., & Purwiyanto, P. (2022). Rancang Bangun Monitoring Akuarium Dan Pakan Ikan Otomatis Berbasis *Internet Of Things* (IOT). *Journal of Electrical Engineering and Computer (JEECOM)*, *4*(2), 62-68. https://ejournal.unuja.ac.id/index.php/jeecom/article/view/4309

Chaidir, A. R., Hidayatullah, A. S., Utomo, S. B., Cahyadi, W., Muldayani, W., Arifin, S., & Wicaksono, I. (2024). Evaluasi Pengujian Alat Pemberi Pakan Ikan Otomatis Berbasis IoT dengan Protokol MQTT. *Jurnal Telematika*, *19*(1), 1-5. https://journal.ithb.ac.id/index.php/telematika/article/view/624

clipse Mosquitto. (2023). *Mosquitto MQTT Broker*. Diakses dari: https://mosquitto.org

Espressif Systems. (2023). *ESP32 Technical Reference Manual*. Espressif Systems.

Koromari, B. I., & David, F. (2023). Perancangan Dan Implementasi Sistem Pakan Otomatis Dan Monitoring Tds Pada Akuarium Ikan Hias Berbasis Iot. IT-Explore: Jurnal Penerapan Teknologi Informasi dan Komunikasi, 2(2), 154-164. https://ejournal.uksw.edu/itexplore/article/view/8903

Ma'shumah, S., Pramarthaningthyas, E. K., & Rohman, F. (2024). Sistem Monitoring Pemberian Pakan Ikan Di Aquarium Ikan Hias Menggunakan Aplikasi Blynk Dengan Memanfaatkan Teknologi Iot. *Uranus : Jurnal Ilmiah Teknik Elektro, Sains dan Informatika*, *2*(3). https://doi.org/10.61132/uranus.v2i3.194

Mischianti, M. (n.d.). *DOIT ESP32 Dev Kit V1 High-Resolution Pinout and Specs*. https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/

Nurhidayah, T., Ulfah, M., & Jamal, N. (2024). Sistem Monitoring Kualitas Air Dan Pakan Otomatis Budidaya Ikan Lele Berbasis *Internet Of Things***.** Jurnal Fokus

Elektroda : Energi Listrik, Telekomunikasi, Komputer, Elektronika dan Kendali, *9*(2). https://doi.org/10.33772/jfe.v9i2.174

PubSubClient Library. (2023). Arduino MQTT Library Documentation. Diakses dari: https://pubsubclient.knolleary.net/

Wijaya, P., & Wellem, T. (2022). Perancangan dan Implementasi Sistem Pemantauan Suhu dan Ketinggian Air pada Akuarium Ikan Hias berbasis IoT. *Jurnal Sistem Komputer Dan Informatika (JSON)*, *4*(1), 225–233. https://doi.org/10.30865/json.v4i1.4539

Zenius. (2021, 8 April). Belajar Rumus Frekuensi Gelombang – Materi Fisika Kelas 11. Diakses dari https://www.zenius.net/blog/rumus-frekuensi/