REAL TIME SYSTEM AND INTERNET OF THINGS FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

SMART FARMING SYSTEM

GROUP 13

AZRA NABILA AZZAHRA	**2306161782**
MUHAMAD REY KAFAKA FADLAN	**2306250573**
MUTIA CASELLA	**2306202870**
WILMAN SARAGIH SITIO	**2306161776**

PREFACE

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga kami dapat menyelesaikan laporan proyek akhir ini dengan baik. Laporan yang merupakan salah satu syarat untuk proyek akhir mata kuliah Sistem Waktu Nyata dan IoT ini bertujuan untuk mendesain, mengimplementasikan, serta menguji perangkat Smart Farming System yang memanfaatkan mikrokontroler ESP32 dengan FreeRTOS.

Kami mengucapkan terima kasih kepada Bapak Fransiskus Astha Ekadiyanto, selaku dosen pengampu mata kuliah Sistem Waktu Nyata dan IoT, serta Kak Phoebe Ivana yang telah memberikan bimbingan dan arahan selama proses pengerjaan proyek akhir ini. Kami juga ingin menyampaikan apresiasi kepada rekan satu kelompok atas kerja sama yang baik sehingga proyek ini dapat diselesaikan tepat waktu.

Kami menyadari bahwa laporan ini masih memiliki kekurangan, sehingga kami sangat terbuka terhadap kritik dan saran yang membangun. Akhir kata, kami berharap laporan ini dapat memberikan manfaat bagi pembaca, khususnya dalam memahami penerapan konsep sistem waktu nyata seperti Task Management, Priority Scheduling, serta integrasi IoT menggunakan platform seperti Blynk untuk memberikan solusi atas permasalahan otomatisasi pertanian pintar pada kehidupan sehari-hari.

Depok, December 8, 2025

Group 13

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Dalam sektor pertanian tradisional, efisiensi penggunaan sumber daya seperti air dan tenaga kerja masih menjadi tantangan utama. Petani biasanya melakukan penyiraman tanaman secara manual berdasarkan perkiraan atau jadwal rutin tanpa mempertimbangkan kondisi aktual dari tanah dan lingkungan. Hal tersebut dapat menimbulkan beberapa masalah serius yang berdampak langsung pada efisiensi dan hasil panen.

Penyiraman yang tidak tepat waktu dan berlebihan dapat mengakibatkan pemborosan air serta menyebabkan kerusakan tanaman akibat kelembaban berlebih. Kurangnya sistem monitoring real-time terhadap parameter lingkungan seperti suhu, kelembaban udara, dan kelembaban tanah juga membuat petani kesulitan dalam mengambil keputusan yang tepat waktu untuk mencegah kerusakan tanaman. Sistem irigasi konvensional memerlukan kehadiran langsung sang petani di lokasi, sehingga akan kurang fleksibel dan dapat meningkatkan beban kerja.

Selain itu, kualitas udara di area pertanian juga dapat memengaruhi kesehatan tanaman. Akan tetapi, hal tersebut seringkali diabaikan dalam sistem pertanian tradisional. Dengan semakin berkembangnya teknologi IoT (*Internet of Things*) dan sistem *embedded*, diperlukan solusi yang dapat mengotomatisasi proses monitoring dan kontrol dalam pertanian untuk meningkatkan efisiensi, mengurangi pemborosan sumber daya, dan memungkinkan pengelolaan jarak jauh.

1.2 PROPOSED SOLUTION

Untuk mengatasi permasalahan tersebut, dikembangkan "*Smart Farming System*" berbasis ESP32 dengan implementasi sistem operasi FreeRTOS. Sistem ini dirancang untuk mengotomatisasi proses irigasi dan *monitoring* kondisi lingkungan pertanian secara *real-time*, sehingga mengubah alur dan cara kerja pertanian tradisional menjadi suatu sistem agrikultur yang terdigitalisasi dan berkelanjutan.

Sistem akan menggunakan beberapa jenis sensor untuk memantau parameter lingkungan seperti suhu, kelembaban udara, kelembaban tanah, dan kualitas udara. Data dari sensor-sensor tersebut kemudian diproses oleh ESP32 yang menjalankan FreeRTOS dengan implementasi *task management*, *queue communication*, dan mutex *synchronization* untuk memastikan operasi *real-time* dapat berjalan dengan baik. Sistem irigasi akan diaktifkan secara otomatis berdasarkan tiga kondisi, yaitu secara periodik berdasarkan interval waktu yang ditentukan, ketika kelembaban tanah turun di bawah *threshold* tertentu, atau secara manual melalui tombol fisik atau aplikasi Blynk. Sistem juga akan memberikan peringatan otomatis melalui platform Blynk ketika parameter lingkungan melebihi batas aman, seperti suhu terlalu tinggi, kelembaban udara terlalu rendah, atau kualitas udara buruk.

Dengan menggunakan pendekatan sistem yang terdistribusi ini, petani dapat mengoptimalkan penggunaan air, mengurangi ketergantungan pada tenaga kerja manual, dan mendapatkan akses *monitoring* secara r*eal-time* dari jarak jauh melalui *smartphone*. Solusi ini dapat meningkatkan efisiensi operasional sekaligus mendukung praktik pertanian berkelanjutan melalui konservasi sumber daya.

1.3 ACCEPTANCE CRITERIA

Kriteria penerimaan untuk proyek ini adalah sebagai berikut:

1. Sistem harus mengimplementasikan FreeRTOS dengan *task-task* paralel yang mengaplikasikan konsep *task management, queue communication,* mutex *synchronization, software timer,* dan *bluetooth low energy.*
2. Sistem harus terintegrasi dengan platform Blynk untuk komunikasi dua arah sebagai perangkat eksternal
3. Sistem harus mampu membaca input *real-time* dari tiga sensor fisik (DHT11, *soil moisture sensor*, MQ-2) dan menampilkan data melalui *Serial Monitor* dan Blynk *dashboard*
4. Sistem harus dapat mengontrol aktuator pompa irigasi berdasarkan tiga kondisi, yaitu *timer interval, threshold* kelembaban tanah, dan input manual dari tombol fisik atau Blynk
5. Sistem harus dapat didemonstrasikan bekerja pada rangkaian fisik ESP32 dengan semua komponen berfungsi sesuai desain
6. Seluruh kode program harus ditulis dengan bahasa C/C++ dengan menggunakan Arduino IDE dan *library* yang telah diajarkan

1.4 ROLES AND RESPONSIBILITIES

Peran dan tanggung jawab yang ditetapkan untuk setiap anggota kelompok adalah sebagai berikut:

Roles	Responsibilities	Person
Role 1	Rangkaian Fisik, Laporan, PPT, README	Muhamad Rey Kafaka Fadlan
Role 2	*Source Code*, Rangkaian Fisik, Laporan	Mutia Casella
Role 3	*Source Code*, Rangkaian Fisik, PPT, README	Wilman Saragih Sitio
Role 4	*Source Code*, Rangkaian Fisik, Laporan	Azra Nabila Azzahra

Table 1. Roles and Responsibilities

1.5 TIMELINE AND MILESTONES

Proyek Smart Farming System akan dilaksanakan selama periode 22 November hingga 8 Desember 2025, di mana rinciannya dapat dilihat pada Gantt Chart berikut.

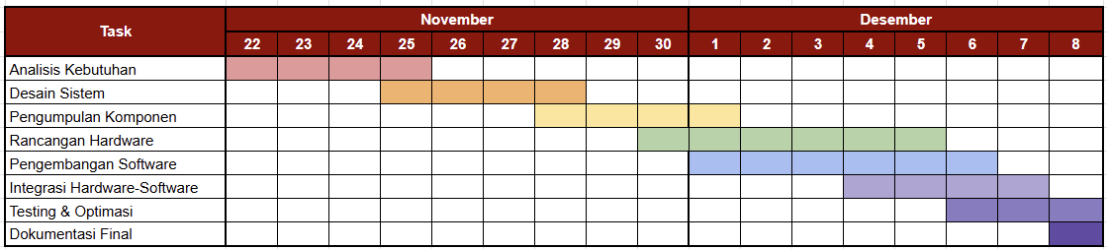


Fig 1. Timeline and Milestones

Milestone Utama:

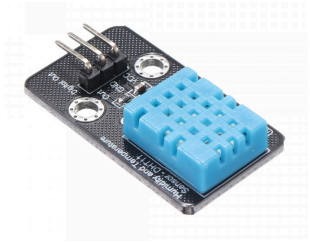
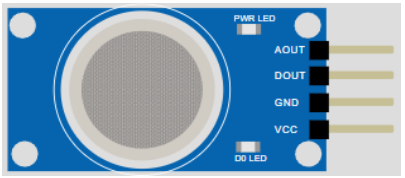
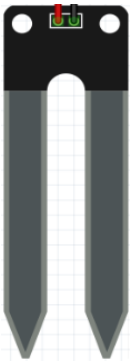
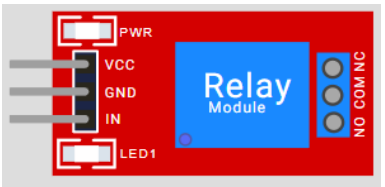
a) **Hardware Design Completion:** 5 Desember 2025 - Finalisasi desain *hardware* termasuk *schematic* rangkaian lengkap untuk sistem *embedded*.
b) **Software Development:** 2 Desember 2025 - Mulai pengembangan *software* dengan implementasi FreeRTOS.
c) **Integration and Testing:** 4 Desember 2025 - Integrasi *hardware* dan *software* dengan *testing* fungsi irigasi otomatis dan komunikasi sensor.
d) **Final Product Assembly and Testing:** 6 Desember 2025 - Produk akhir dirakit lengkap dan dilakukan *final testing* untuk verifikasi *acceptance criteria*.

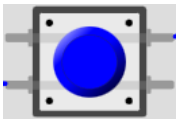
CHAPTER 2

IMPLEMENTATION

2.1 HARDWARE DESIGN AND SCHEMATIC

Hardware untuk *Smart Farming System* dirancang dengan mengintegrasikan mikrokontroler ESP32 dengan tiga sensor input, satu aktuator output, serta *interface* kontrol manual. Desain ini mengutamakan keandalan pembacaan sensor, serta efisiensi dan responsivitas dari sistem kontrol *real-time* untuk operasi sistem pertanian autonom. Berikut komponen-komponen *hardware* yang akan digunakan pada proyek ini, yaitu:

Komponen		Fungsi
DHT11		Sensor suhu dan kelembapan udara
MQ-2		Sensor yang akan memeriksa kualitas udara
Soil Moisture Sensor		Sensor untuk mengukur kelembapan tanah
Relay Module		Mengendalikan pompa untuk irigasi

Button		Kontrol penyiraman manual secara fisik
ESP32		Menjalankan FreeRTOS dan mengontrol semua logika sistem

Rangkaian lengkap sistem (*schematic*) dapat dilihat pada **Appendix A**. Berikut adalah penjelasan koneksi setiap modul:

1. **Modul ESP32:** berperan sebagai pusat kendali dan pemrosesan utama, dengan *power supply* diberikan melalui pin VIN (5V) atau 3V3, serta *return path* melalui pin GND.
2. **Sensor DHT11:** menggunakan protokol komunikasi *one-wire*, dengan pin VCC terhubung ke 3.3V ESP32 dan pin GND ke GND ESP32. Pin DATA yang terkonfigurasi sebagai `DHTPIN` pada kode terhubung ke pin GPIO 15 pada ESP32.
3. **Sensor Soil Moisture:** pin VCC terhubung ke 3.3V ESP32 dan pin GND ke GND ESP32. Pin AO yang terkonfigurasi sebagai `SOIL_PIN` pada kode terhubung ke pin GPIO 35 pada ESP32.
4. **Sensor Gas MQ-2:** pin VCC terhubung ke 5V (pin VIN ESP32) dan pin GND ke GND ESP32. Pin AO yang terkonfigurasi sebagai `MQ_PIN` pada kode terhubung ke pin GPIO 34 pada ESP32 untuk membaca nilai ADC.
5. **Sistem Irigasi:** pin GPIO 5 yang dikonfigurasi sebagai `PUMP_LED` pada kode terhubung ke *relay module* sebagai sinyal kontrol. Pin NO pada *relay* mengalirkan daya ke pompa air (*pump*) untuk aktivasi irigasi.
6. **Tombol Kontrol Manual:** salah satu kaki *push button* terhubung ke pin GPIO 26 (`LOCAL_BUTTON`) dengan konfigurasi internal *pull-up*, dan kaki lainnya ke GND ESP32, menghasilkan logika *active-low* (LOW saat ditekan).

Desain *hardware* ini memetakan semua kebutuhan fungsional sistem ke dalam rangkaian elektronik yang praktis. Setiap keterhubungannya telah dirancang untuk mendukung implementasi *software real-time* berbasis FreeRTOS, dengan pin GPIO yang sudah sesuai dengan konfigurasi pada kode program.

2.2 SOFTWARE DEVELOPMENT

2.2.1. SYSTEM ARCHITECTURE OVERVIEW

Pada pengembangannya, *software* dikembangkan menggunakan ESP32 sebagai mikrokontroler utama yang mengelola proses pembacaan sensor, pengendalian irigasi, notifikasi peringatan lingkungan, serta komunikasi dengan aplikasi Blynk dan BLE (*Bluetooth Low Energy*). Program ini dirancang dengan menggunakan FreeRTOS untuk memanfaatkan *task scheduling*, sehingga setiap proses dapat berjalan secara paralel tanpa saling mengganggu antar *task*. Sistem ini juga terdiri dari beberapa *task* yang bertugas untuk melakukan pembacaan sensor, kontrol aktuator, pemrosesan perintah irigasi, membaca input tombol lokal, dan mengirimkan data ke server Blynk serta BLE.

Sistem dimulai dengan inisialisasi komponen *hardware* dan jaringan. Setelah semua komponen siap, FreeRTOS menjalankan lima *task* secara paralel. *Sensor Task* membaca data dari sensor DHT11, *Soil Moisture*, dan MQ-2 setiap 1 detik dan mengirimkannya ke *Control Task*. *Control Task* kemudian memproses data, mengevaluasi peringatan berdasarkan *threshold*, dan memicu simulasi irigasi. *Software timer* berjalan secara periodik setiap 10 detik untuk irigasi otomatis berbasis waktu. *Button Task* memantau input fisik, *Blynk Task* mengelola komunikasi cloud, dan *BLE Task* menyediakan *interface* monitoring lokal melalui *Bluetooth*. Semua perintah irigasi dari berbagai sumber dikumpulkan dalam `irrigationQueue` yang diproses oleh *Control Task*, yang mana kemudian menjalankan fungsi irigasi dengan proteksi mutex.

2.2.2. COMPONENT IMPLEMENTATION DETAIL

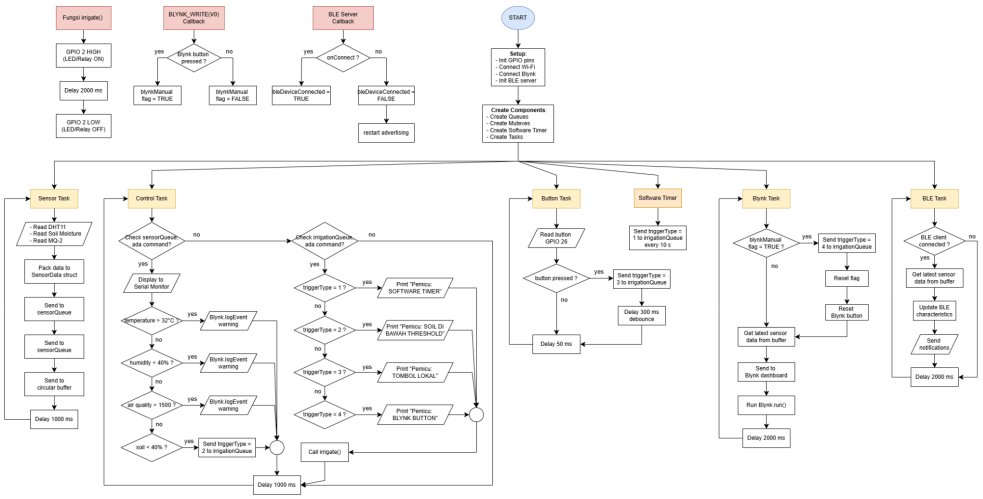


Fig 2. Flowchart

1. **Inisialisasi Sistem:**

   Proses *setup* diawali dengan konfigurasi pin GPIO, inisialisasi *serial communication* pada *baudrate* 115200, dan *setup* sensor DHT11. Sistem kemudian menghubungkan ke jaringan Wi-Fi menggunakan kredensial yang telah ditentukan. Setelah Wi-Fi terhubung, koneksi ke server Blynk dilakukan untuk membuka komunikasi IoT *cloud*. Bersamaan dengan proses tersebut, server BLE diinisialisasi dengan nama *device* "SmartFarm_K13" dan empat karakteristik BLE untuk setiap parameter sensor. Terakhir, komponen FreeRTOS dibuat, termasuk *queue* untuk komunikasi antar-*task*, mutex untuk sinkronisasi, *software timer* untuk irigasi periodik, dan kelima *task* yang akan berjalan paralel.

2. **Task Sensor:**

   Task ini berjalan pada *core* 0 dengan prioritas 2 dan periode 1000 ms. Pada setiap siklusnya, *task* membaca data dari tiga sensor, yaitu suhu dan kelembaban udara dari DHT11 menggunakan protokol *one-wire*, kelembaban tanah dari *Soil Moisture Sensor* melalui ADC GPIO 35 yang dikonversi ke persentase 0-100%, dan kualitas udara dari MQ-2 melalui ADC GPIO 34 dengan rentang 0-4095. Data sensor diproses dalam struktur `SensorData` dan dikirim ke `ControlTask` melalui `sensorQueue`. Data juga disimpan dalam *circular buffer* berukuran 5 untuk digunakan oleh `BlynkTask` dan `BLETask`, memastikan data terbaru selalu tersedia untuk *monitoring*. Sistem menggunakan dua struktur data utama, yaitu `SensorData` dan `IrrigationCommand` dengan `triggerType` 1 untuk *software timer*, 2 untuk *soil threshold*, 3 untuk tombol lokal, dan 4 untuk Blynk *button*.

3. **Task Kontrol Irigasi:**

   Berjalan pada *core* 0 dengan prioritas 3 dan periode 1000 ms, *task* ini berfungsi untuk menerima dan memproses data dari `sensorQueue`. Data ditampilkan di *Serial Monitor* untuk *debugging real-time*. Sistem evaluasi kondisi lingkungan akan memeriksa tiga *threshold* yang ditentukan berdasarkan kebutuhan tanaman, yaitu suhu di atas 32°C (batas *heat stress*), kelembaban udara di bawah 40% (mencegah dehidrasi tanaman), dan kualitas udara melebihi 1500 ADC (indikator polusi udara sedang). Jika kondisi terpenuhi, notifikasi akan dikirimkan ke Blynk. Untuk kontrol irigasi, *task*

memeriksa apakah kelembaban tanah kurang dari 40% dan akan mengirim perintah irigasi jika diperlukan. *Task* juga memproses semua perintah irigasi dari `irrigationQueue` yang berasal dari berbagai sumber.

4. **Software Timer untuk Irigasi:**

   Dengan FreeRTOS *Software Timer* API, sistem membuat *timer* periodik bernama `IrrigationTimer` dengan periode 10 detik (untuk mode simulasi). *Timer* ini berjalan secara otomatis dan memanggil *callback function* `irrigationTimerCallback()` setiap periode habis. *Callback function* mengirim perintah irigasi dengan `triggerType` 1 (*software timer*) ke `irrigationQueue`. Metode ini lebih efisien dibandingkan *polling* menggunakan `millis()` karena bersifat *event-driven* dan terintegrasi dengan *scheduler* FreeRTOS, sehingga mengurangi *overhead* CPU.

5. **Task Tombol:**

   Berjalan pada *core* 1 dengan prioritas 1 dan periode 50 ms, *task* ini khusus menangani input fisik dari *push button* di GPIO 26. *Task* melakukan *debouncing software* dengan mendeteksi *falling edge* (transisi HIGH ke LOW) pada sinyal *button*. Ketika tombol ditekan, *task* mengirim perintah irigasi dengan `triggerType` 3 (*local button*) ke `irrigationQueue`. *Delay* 300 ms diterapkan untuk mencegah *bouncing*. Prioritas rendah (1) dipilih karena input manual tidak memerlukan *respons time* yang ketat dibandingkan kontrol sensor.

6. **Task Blynk:**

   Berjalan pada *core* 1 dengan prioritas 2 dan periode 2000 ms, *task* ini mengelola komunikasi *output* ke platform Blynk. *Task* memproses *flag* `blynkManual` yang diaktifkan oleh *callback* `BLYNK_WRITE(V0)` saat tombol virtual ditekan di aplikasi, kemudian mengirim perintah irigasi dengan `triggerType` 4 ke `irrigationQueue`. Data sensor dari *circular buffer* dikirim ke *virtual pins* V1-V4 untuk *update dashboard*. *Task* juga menjalankan `Blynk.run()` untuk menjaga koneksi. Notifikasi peringatan dikirim oleh `ControlTask` melalui pemanggilan `Blynk.logEvent()` ketika kondisi *threshold* terdeteksi.

7. **Task BLE (*Bluetooth Low Energy*):**

Berjalan pada *core* 0 dengan prioritas 2 dan periode 2000 ms, *task* ini mengimplementasikan BLE *Server* dengan empat karakteristik *notifiable*. Server menggunakan UUID *service* khusus dan mendeteksi koneksi melalui *callback*. Ketika *device* BLE *client* terhubung, *task* mengirim data sensor terbaru dalam format string melalui mekanisme BLE *notifications* setiap 2 detik. Server akan secara otomatis *restart advertising* ketika *client* tidak terhubung. Fungsi *task* ini adalah untuk menyediakan layanan *monitoring* lokal tanpa perlu internet.

8. **Aktuasi Irigasi:**

   Fungsi `irrigate()` dipanggil oleh `ControlTask` ketika ada perintah irigasi dari `irrigationQueue`. Fungsi menggunakan `pumpMutex` untuk mencegah *race condition* pada akses ke aktuator pompa jika *multiple trigger* terjadi bersamaan. Dalam demonstrasi proyek ini, fungsi akan mengaktifkan GPIO 5 yang terhubung ke *relay module*. *Relay* kemudian mengontrol pompa air, dengan durasi aktivasi 2 detik menggunakan `vTaskDelay()` untuk menjaga sistem tetap responsif.

2.3 HARDWARE AND SOFTWARE INTEGRATION

Integrasi *hardware* dan *software* dilakukan dengan menghubungkan seluruh komponen fisik sesuai *schematic* pada **Appendix A**, lalu kemudian meng-*upload* dan *run* kode program ke ESP32 melalui Arduino IDE. Proses integrasi bertujuan untuk memastikan semua fungsi sistem dapat beroperasi sesuai dengan desain yang telah dibuat.

2.3.1. PIN MAPPING AND INTERFACE

Pemetaan pin I/O pada mikrokontroler ESP32 merupakan langkah awal dalam proses integrasi. Setiap komponen fisik dihubungkan ke pin GPIO spesifik yang telah didefinisikan sebagai variabel dalam kode program, sehingga memungkinkan komunikasi yang efisien antara *hardware driver* dan *task* FreeRTOS.

Komponen Hardware	Pin ESP32	Variabel Kode	Tipe Sinyal
DHT11	GPIO 15	`DHTPIN`	Digital (One-wire)
Soil Moisture Sensor	GPIO 35	`SOIL_PIN`	Analog (ADC)

MQ-2 Sensor	GPIO 34	`MQ_PIN`	Analog (ADC)
Relay Module	GPIO 5	`PUMP_LED`	Digital Output
Push Button	GPIO 26	`LOCAL_BUTTON`	Digital Input (Pull-up)

2.3.2. SENSOR INTEGRATION FLOW TO FREERTOS TASKS

Integrasi sensor ke dalam *manajemen task* FreeRTOS dilakukan melalui mekanisme *queue* untuk memastikan pemisahan tugas dan operasi *non-blocking*.

1. **Pengambilan Data Fisik:** *Task* sensor (`vTaskSensor`) melakukan *polling* terhadap pin GPIO 15, 35, dan 34 setiap 1000 ms. Untuk sensor DHT11, pembacaan dilakukan dengan menggunakan *driver library*, yang mana aksesnya dilindungi oleh mutex (`sensorMutex`) untuk menjaga integritas pembacaan. Di sisi lain, untuk sensor MQ-2 dan *Soil Moisture*, data mentah diperoleh melalui fungsi `analogRead()` pada pin ADC.
2. **Pemrosesan Software:** Data mentah yang diperoleh kemudian diolah dan diformat ke dalam *struct* `SensorData` agar siap digunakan oleh logika kontrol. Pemrosesan ini melibatkan proses konversi nilai. Nilai *float* suhu dan kelembaban yang sudah diukur oleh *library* DHT dapat dimasukkan langsung. Nilai ADC (0-4095) dari *Soil Moisture* dipetakan secara linear menjadi persentase kelembaban 0-100% menggunakan fungsi `map()` dan `constrain()`. Nilai ADC mentah dari sensor gas MQ-2 juga dimasukkan ke dalam *struct* sebagai indikator kualitas udara.
3. **Transmisi Data:** Setelah data sensor sudah diproses dan dikemas dalam *struct* `SensorData`, data tersebut kemudian didistribusikan ke dua jalur. Jalur pertama yaitu dikirim ke *task* kontrol (`vControlTask`) melalui `sensorQueue`. Jalur kedua, salinan data sensor terbaru akan disimpan ke *circular buffer* (`sensorBuffer`) di *shared memory*, yang mana berfungsi sebagai sumber data *real-time* bagi *task* Blynk dan *task* BLE untuk keperluan *monitoring*.

2.3.3. CONTROL TASK INTEGRATION TO IRRIGATION ACTUATOR

Inti dari sistem *real-time* ini adalah kontrol untuk aktuator irigasi, yang mana melibatkan pengambilan keputusan dari berbagai sumber input yang terdistribusi dan membutuhkan mekanisme sinkronisasi yang tepat. (wait tdr bentar)

1. **Pengumpulan Perintah:** *Task* kontrol (`vTaskControl`) memproses data kelembaban tanah dari sensorQueue. Jika kelembaban kurang dari 40%, perintah irigasi dikirim ke `irrigationQueue` (*Trigger Type* 2). *Task* ini juga menerima perintah dari sumber pemicu lainnya (*Software Timer*, Tombol Lokal, dan Blynk).
2. **Aksi Irigasi:** Setiap perintah dari `irrigationQueue` memicu pemanggilan fungsi `irrigate()`.
3. **Sinkronisasi Aktuator:** Fungsi `irrigate()` dilindungi oleh mutex `pumpMutex`. Sebelum mengaktifkan pompa pada GPIO 5, fungsi harus berhasil memperoleh mutex tersebut. Mekanisme ini mencegah kondisi balapan (race condition) yang dapat terjadi jika perintah irigasi dari sumber berbeda (misalnya *Timer* dan Tombol Lokal) diterima secara bersamaan, memastikan integritas kontrol aktuator.
4. **Aktuasi Fisik:** Setelah mutex diperoleh, pin GPIO 5 (`PUMP_LED`) diatur ke logika HIGH selama 2 detik untuk menyimulasikan operasi pompa air, kemudian diatur kembali ke LOW, dan mutex dilepaskan.

2.3.4. MANUAL INPUT TASK INTEGRATION

1. Integrasi Hardware: Tombol dihubungkan antara GPIO 26 dan GND, dengan pin dikonfigurasi sebagai *Digital Input* dengan internal *pull-up* untuk menghasilkan logika *active-low*.
2. **Task Implementation:** *Task* tombol (`vTaskButton`) berjalan dengan prioritas rendah dan periode *polling* yang cepat (50 ms) untuk memastikan deteksi penekanan tombol yang responsif.
3. **Software Debouncing:** *Task* ini mengimplementasikan logika *software debouncing* untuk mengatasi *bouncing* fisik pada tombol. Setelah mendeteksi *falling edge* yang valid, task mengirimkan perintah irigasi (*Trigger Type* 3) ke `irrigationQueue`.

2.3.5. CLOUD AND LOCAL INTEGRATION

Integrasi konektivitas memungkinkan komunikasi dua arah dengan perangkat eksternal, memanfaatkan kemampuan Wi-Fi dan Bluetooth hardware ESP32.

- **Blynk (IoT Cloud):**
  1. *Task* Blynk (`vTaskBlynk`) menjalankan `Blynk.run()` untuk menjaga koneksi Wi-Fi yang stabil.
  2. Perintah irigasi dari aplikasi *smartphone* diterima melalui *Virtual Pin Callback* (`BLYNK_WRITE(V0)`), yang kemudian mengirim perintah irigasi (*Trigger Type* 4) ke `irrigationQueue`.
  3. Data sensor terbaru diambil dari *circular buffer* di *shared memory* dan dikirim ke *virtual pins* Blynk (V1-V4), yang memungkinkan *monitoring real-time* kondisi pertanian dari jarak jauh.
- **BLE (Bluetooth Low Energy):**
  1. Task BLE (`vTaskBLE`) mengaktifkan server Bluetooth pada *hardware* ESP32.
  2. Data sensor dikirim secara lokal (tanpa internet) melalui mekanisme BLE *Notifications* yang terintegrasi dengan data terbaru dari *circular buffer*, menyediakan jalur monitoring cadangan atau untuk lingkungan tanpa koneksi Wi-Fi.

CHAPTER 3

TESTING AND EVALUATION

3.1 TESTING

Pada fase pengujian Smart Farm, langkah pertama adalah menghubungkan ke semua sensor, diantaranya *soil moisture sensor*, MQ-2 Sensor, dan DHT11, ke mikrokontroler ESP32. Dengan menggunakan *Serial Monitor*, kita mampu mengecek pembacaan semua sensor untuk memastikan *output* yang dihasilkan akurat dalam merepresentasikan kondisi dunia nyata. Langkah berikutnya adalah menguji konektivitas Wi-Fi dan Bluetooth, serta integrasi platform Blynk. Hal ini mencakup verifikasi bahwa sistem dapat terhubung ke jaringan Wi-Fi dan mampu mengirim data ke aplikasi Blynk dan Bluetooth secara stabil. Aplikasi Blynk dan Bluetooth juga diuji kemampuannya untuk menerima data sensor secara *real-time* dan menerima input pengguna. Skenario pengujian ini dibagi menjadi tiga kategori utama, yaitu:

a. Inti Logika Sistem

- Memastikan kelima *task* (`vSensorTask`, `vControlTask`, `vTaskBlynk`, `vTaskButton`, `vTaskBLE`) berjalan secara konkuren di *core* 0 dan *core* 1 sesuai prioritas yang ditetapkan, dapat dilihat dari log *Serial Monitor*.
- Menguji akurasi pembacaan sensor DHT11 untuk suhu dan kelembaban udara, memverifikasi konversi ADC *Soil Moisture* pada kondisi kering dan basah, dan mengecek respons MQ-2 terhadap simulasi udara tidak bersih.
- Menguji *debouncing* pada `vButtonTask` dengan menekan tombol `LOCAL_BUTTON` (GPIO 26) secara cepat. Perintah irigasi (*Trigger Type* 3) harus terpicu hanya sekali per penekanan.

b. Kontrol dan Sinkronisasi Aktuator

- Menguji respons `vControlTask` ketika kelembaban tanah turun di bawah *threshold* 40%, *task* harus segera mengirim perintah irigasi (*Trigger Type* 2) ke `irrigationQueue`.
- Memverifikasi `irrigationTimer` (*software timer*) berhasil terpicu setiap 10 detik secara periodik, dan mengirim perintah irigasi (*Trigger Type* 1) ke `irrigationQueue`.

- Menguji apakah `vControlTask` memicu fungsi `Blynk.logEvent()` ketika salah satu *threshold* peringatan melewati batas yang telah ditentukan.
- Melakukan tes pada `pumpMutex` dengan memicu irigasi dari dua sumber berbeda secara bersamaan. Pompa harus mengantri, di mana hanya satu pemicu irigasi saja yang dapat berjalan dalam satu waktu.

c. Konektivitas Cloud dan Lokal

- Memastikan bahwa koneksi Wi-Fi dan Blynk stabil, task `vBlynkTask` berhasil menjalankan `Blynk.run()` dan memperbarui data sensor terbaru dari `sensorBuffer` ke *virtual pin* V1-V4 di *dashboard* Blynk.
- Menguji respons *callback* `BLYNK_WRITE(V0)` saat tombol Blynk ditekan, di mana perintah irigasi (*Trigger Type* 4) seharusnya diteruskan ke `irrigationQueue`.
- Memverifikasi server BLE "SmartFarm_K13" dapat ditemukan dan dihubungkan oleh perangkat lain. Task `vBLETask` harus dapat mengirimkan notifikasi data sensor secara periodik.

3.2 RESULT

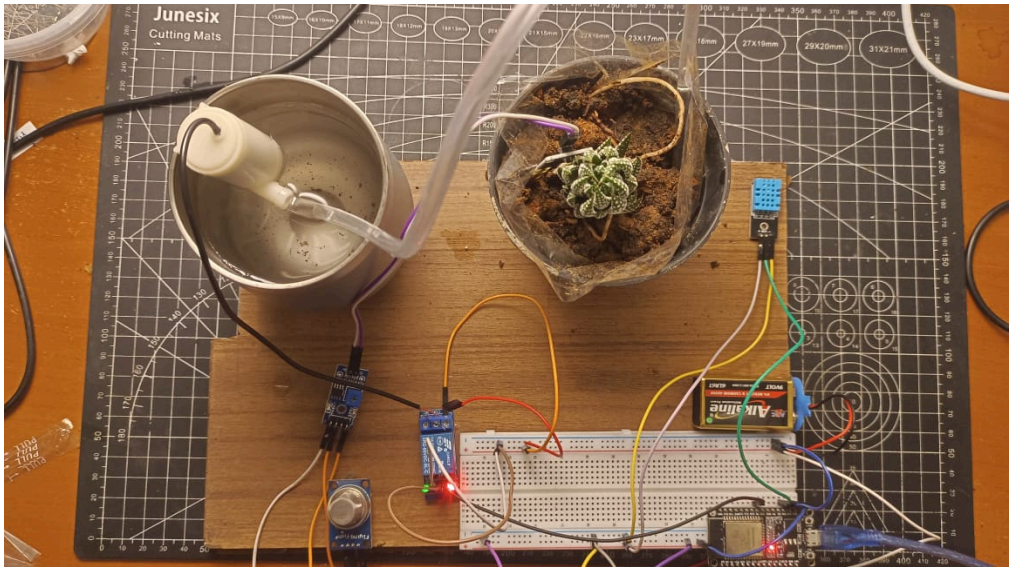


Fig 3.2.1 Physical Testing Result

Hasil dari fase pengujian mengindikasikan bahwa sistem bekerja seperti yang diinginkan dengan pembacaan sensor, koneksi, dan penerimaan input berfungsi tanpa *error*

dan merespons sesuai parameter yang telah ditetapkan. Pada Fig 3.2.2, kita bisa melihat bahwa rangkaian dapat menunjukkan output dan menggunakan *internal timer* untuk menentukan kapan irigasi dilakukan. Dalam pengujian terkontrol, saat kelembaban tanah berada di bawah *threshold*, seperti yang ditunjukkan pada Fig 3.2.3, rangkaian mampu merespons dan irigasi dijalankan.

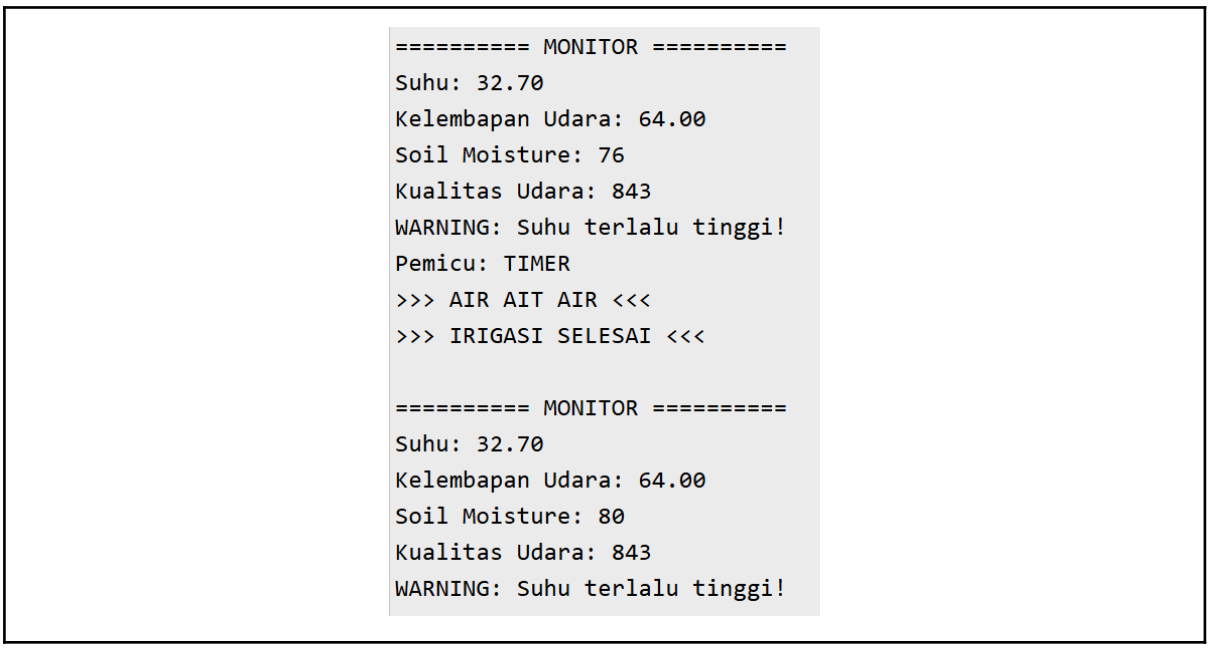


Fig 3.2.2 Serial Monitor Output

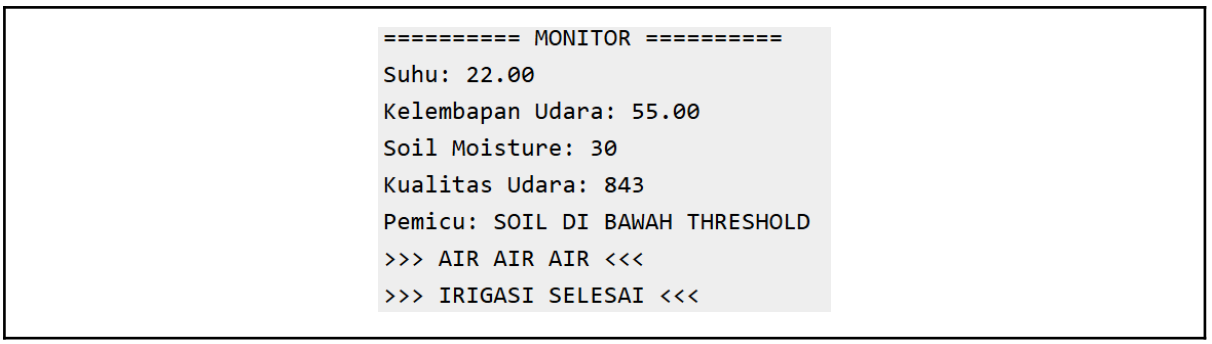


Fig 3.2.3 Serial Monitor saat Salah Satu Parameter Ter-*trigger*

Konektivitas dan integrasi ke platform Blynk juga berhasil dilakukan, seperti yang terlihat pada Fig 3.2.4. *Interface* yang disediakan oleh Blynk memungkinkan pengguna untuk melihat pembacaan sensor secara *real-time* dan mengatur proses irigasi secara manual dari jauh.

Kami menggunakan tampilan gauge yang memberikan representasi visual dan numerik. Setiap gauge dikonfigurasikan untuk menunjukkan pembacaan saat ini dari sensor tertentu, seperti kelembaban tanah, suhu dan kelembaban udara, dan kebersihan udara. Selain itu, kami menambahkan *switch* yang di-set *bounce* untuk melakukan irigasi manual.

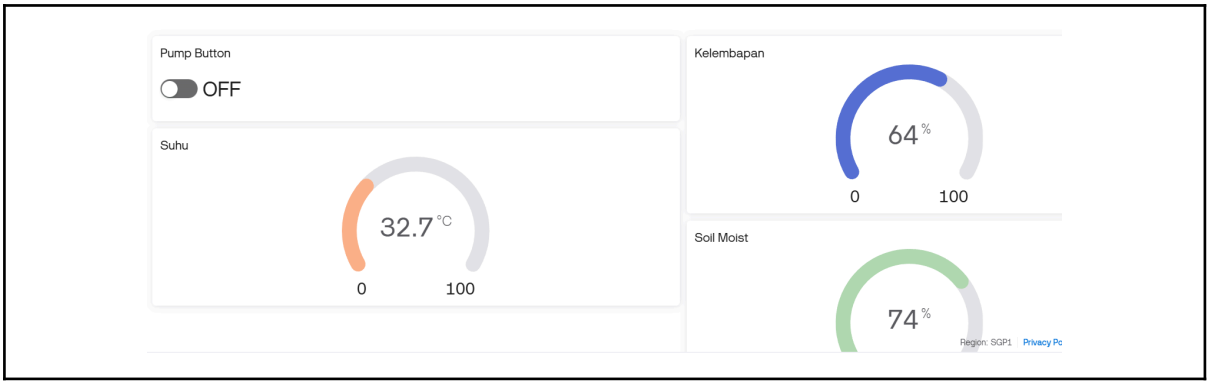


Fig 3.2.4 Blynk Output

Dalam perakitan fisik yang bisa dilihat pada Fig 3.2.1, kami menyatukan semua komponen dalam papan kayu yang menghubungkan papan breadboard, sensor, pot tanaman, dan sumber air. Breadboard di sini digunakan untuk perakitan dan menyediakan platform untuk memusatkan sumber listrik. Relay digunakan untuk mengontrol keaktifan *water pump*. Hasil rakitan menunjukkan respons yang seragam dengan output *Serial Monitor*.

3.3 EVALUATION

Selama pengujian sistem *Smart Farm*, integrasi sensor suhu dan kelembaban dengan ESP32 telah diverifikasi keakuratannya melalui Serial Monitor. Konektivitas Wi-Fi, Bluetooth dan integrasi Blynk juga telah diuji, mengkonfirmasi transmisi data real-time yang akurat dan responsif terhadap input pengguna.

Namun, perlu dicatat bahwa pompa memiliki ketidakstabilan dalam kerjanya. Hal ini disebabkan dari masalah *wiring* dimana penghubungan dua kabel dilakukan dengan menggunakan *Electric Tape*, yang membuatnya rentan terlepas.

Selain itu, pompa sendiri terkadang memiliki masalah dalam memompa air, sehingga pelepasan selang input harus disingkirkan terlebih dahulu.

Dari sisi sensor, MQ-2 terkadang lambat dan/atau gagal memberikan bacaan yang akurat. Pengujian simulasi menggunakan Wokwi menunjukkan bahwa masalah ini bersifat spesifik pada hardware.

Dalam pengembangannya, penambahan sensor dan fitur dapat dilakukan untuk mendukung pengotomatisan lebih lanjut. Hal ini bisa mencangkup ditambahkannya Ultrasonic sensor untuk mendeteksi sisa air dalam tangki atau pendeteksian cahaya untuk mengatur intensitas cahaya buatan.

Dengan evaluasi ini, sistem dapat lebih dikembangkan untuk pengefisiensian sumber daya dan membantu otomatisasi pertanian berkelanjutan.

CHAPTER 4

CONCLUSION

Proyek *Smart Farming System* yang telah dikembangkan berhasil membuktikan bahwa penerapan IoT dapat meningkatkan efisiensi dan otomasi dalam pertanian modern. Sistem mampu melakukan monitoring kondisi lingkungan pertanian secara *real-time* melalui sensor seperti DHT11, MQ-2 Gas, dan *Soil Moisture*, serta mengintegrasikan pembacaan tersebut ke Blynk dan koneksi BLE untuk akses monitoring lokal.

Selain itu, sistem irigasi otomatis juga dapat bekerja sesuai desain, yaitu aktif berdasarkan tiga mekanisme pemicu seperti interval waktu menggunakan *software timer*, pendeteksian kelembaban tanah di bawah *threshold*, dan input manual melalui tombol fisik maupun aplikasi Blynk. Penggunaan FreeRTOS dengan *task scheduling*, *queue communication*, dan mutex *synchronization* terbukti meningkatkan responsivitas, keandalan, dan efisiensi kontrol aktuator sehingga tidak terjadi konflik antara berbagai sumber pemicu irigasi.

Dari hasill pengujian, semua komponen dapat terintegrasi dengan baik. Hal ini menunjukkan bahwa pemangaatan *real-time* dan IoT dapat memberikan solusi yang efektif dalam otomasi dalam pertanian modern, terutama pada optimasi penggunaan air dan monitoring lingkungan secara berkelanjutan.

Untuk ke depannya, sistem kemungkinan masih bisa untuk dikembangkan seperti implementasi solar panel untuk energi mandiri, penambahan sensor pH tanah, sistem penyimpanan data, dan lainnya untuk prediksi kebutuhan irigasi. Dengan pengembangan tersebut, sistem dapat semakin mendukung *smart farming system* untuk kedepannya.

REFERENCES

[1] ESP32 I/O, "ESP32 - Gas Sensor," ESP32 Tutorial, 2025. https://esp32io.com/tutorials/esp32-gas-sensor (accessed Dec. 08, 2025).

[2] Digilab UI, "Module 2 - Task Manage... | Digilab UI," Digilabdte.com, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-2-task-management (accessed Dec. 08, 2025).

[3] Digilab UI, "Module 3 - Memory Mana... | Digilab UI," Digilabdte.com, 2018. https://learn.digilabdte.com/books/internet-of-things/chapter/module-3-memory-management-queue (accessed Dec. 08, 2025).

[4] Digilab UI, "Module 4 - Deadlock & ... | Digilab UI," Digilabdte.com, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-4-deadlock-synchronization (accessed Dec. 08, 2025).

[5] Digilab UI, "Module 5 - Software Timer | Digilab UI," Digilabdte.com, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-5-software-timer (accessed Dec. 08, 2025).

[6] Digilab UI, "Module 6 - Bluetooth &... | Digilab UI," *Digilabdte.com*, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-6-bluetooth-ble (accessed Dec. 08, 2025).

[7] Digilab UI, "Module 7 - MQTT, HTTP,... | Digilab UI," Digilabdte.com, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-7-mqtt-http-wifi (accessed Dec. 08, 2025).

[8] Digilab UI, "Module 9 - IoT Platfor... | Digilab UI," Digilabdte.com, 2025. https://learn.digilabdte.com/books/internet-of-things/chapter/module-9-iot-platforms-blynk-and-red-node (accessed Dec. 08, 2025).

[9] Random Nerd Tutorials, "ESP32 FreeRTOS: Software Timers/Timer Interrupts (Arduino) | Random Nerd Tutorials," Random Nerd Tutorials, Nov. 20, 2025. https://randomnerdtutorials.com/esp32-freertos-software-timers-interrupts/ (accessed Dec. 08, 2025).

[10] Random Nerd Tutorials, "ESP32 Bluetooth Low Energy (BLE) on Arduino IDE | Random Nerd Tutorials," Random Nerd Tutorials, May 16, 2019.

https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/ (accessed Dec. 08, 2025).

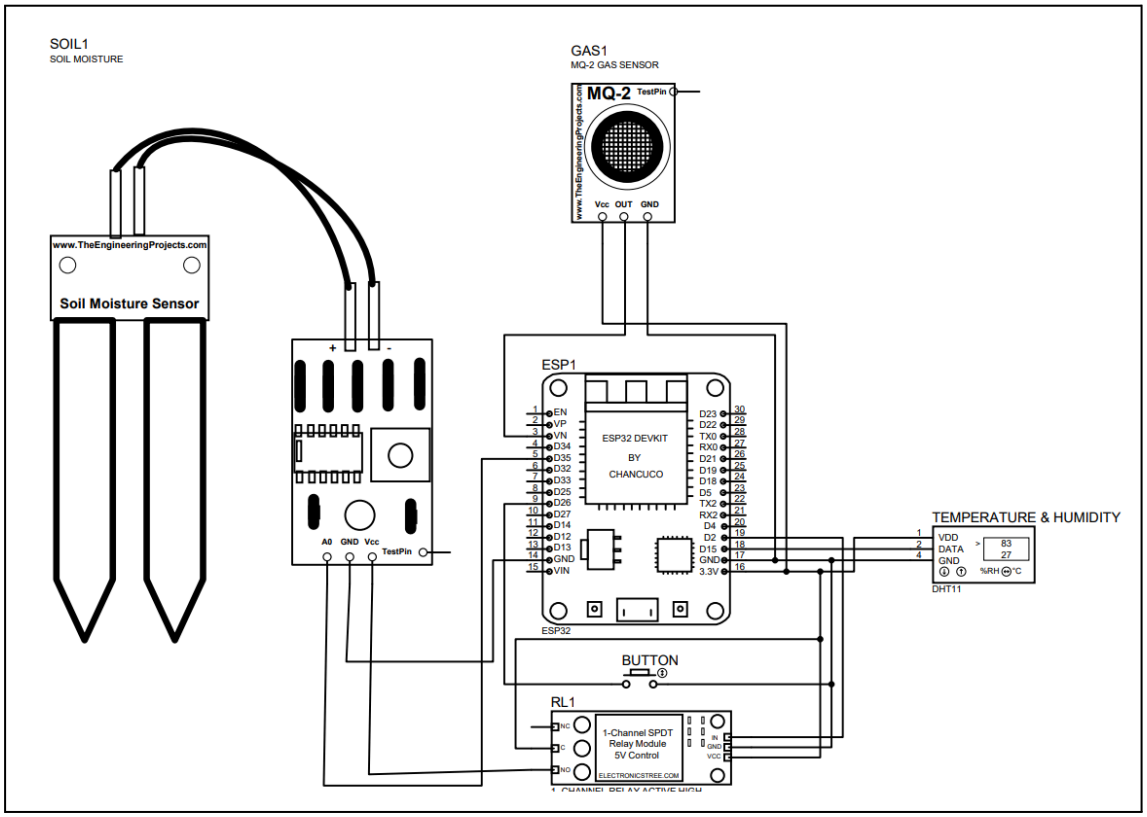
[11] ESP32 I/0, “ESP32 - Soil Moisture Sensor,” ESP32 Tutorial, 2025. https://esp32io.com/tutorials/esp32-soil-moisture-sensor (accessed Dec. 08, 2025).

[12] ESP32 I/O, “ESP32 - Soil Moisture Sensor Pump | ESP32 Tutorial,” ESP32 Tutorial, 2018. https://esp32io.com/tutorials/esp32-soil-moisture-sensor-pump (accessed Dec. 08, 2025).

[13] ESP32 I/O, “ESP32 - Automatic Irrigation System,” ESP32 Tutorial, 2018. https://esp32io.com/tutorials/esp32-automatic-irrigation-system (accessed Dec. 08, 2025).

APPENDICES

Appendix A: Project Schematic



Appendix B: Source Code

```
/*************************************************
 * SMART FARM SYSTEM - Kelompok 13
 * Fitur:
 *   - Penyiraman interval timer (SOFTWARE TIMER)
 *   - Penyiraman berdasarkan soil threshold
 *   - Penyiraman manual dari tombol & Blynk
 *   - Warning suhu, kelembapan, kualitas udara
 *   - BLE Server untuk monitoring sensor
 *************************************************/

#define BLYNK_TEMPLATE_ID "TMPL6iLhzYrwu"
#define BLYNK_TEMPLATE_NAME "ProyekAkhir"
#define BLYNK_AUTH_TOKEN "mgDJ5ZeVSOWgJtP-XtuUHCConoORnOY8"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
```

```
#include <DHT.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <freertos/queue.h>
#include <freertos/semphr.h>
#include <freertos/timers.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// ====== WiFi ======
char ssid[] = "Wi-Fi";
char pass[] = "password";

// ====== Pin Assignment ======
#define DHTPIN 15
#define DHTTYPE DHT11
#define SOIL_PIN 35
#define MQ_PIN 34
#define PUMP_LED 5
#define LOCAL_BUTTON 26

DHT dht(DHTPIN, DHTTYPE);

// ====== Threshold ======
int soilThreshold = 40;
float tempThreshold = 32.0;
float humThreshold = 40.0;
int mqThreshold = 1500;

// ====== Irrigation Settings ======
unsigned long interval = 10000;  // 10 detik untuk testing

// ====== BLE Configuration ======
#define SERVICE_UUID        "12345678-1234-1234-1234-123456789abc"
#define CHAR_TEMP_UUID      "12345678-1234-1234-1234-123456789ab1"
#define CHAR_HUM_UUID       "12345678-1234-1234-1234-123456789ab2"
#define CHAR_SOIL_UUID      "12345678-1234-1234-1234-123456789ab3"
#define CHAR_MQ_UUID        "12345678-1234-1234-1234-123456789ab4"

BLEServer* pServer = NULL;
BLECharacteristic* pCharTemp = NULL;
BLECharacteristic* pCharHum = NULL;
BLECharacteristic* pCharSoil = NULL;
BLECharacteristic* pCharMQ = NULL;
bool bleDeviceConnected = false;

// ====== Task Management ======
TaskHandle_t sensorTaskHandle = NULL;
TaskHandle_t controlTaskHandle = NULL;
TaskHandle_t blynkTaskHandle = NULL;
TaskHandle_t buttonTaskHandle = NULL;
```

```
TaskHandle_t bleTaskHandle = NULL;

// ====== Queue Management ======
QueueHandle_t sensorQueue = NULL;
QueueHandle_t irrigationQueue = NULL;

// ====== Mutex Synchronization ======
SemaphoreHandle_t pumpMutex = NULL;
SemaphoreHandle_t sensorMutex = NULL;
SemaphoreHandle_t blynkMutex = NULL;
SemaphoreHandle_t bleMutex = NULL;

// ====== Software Timer for Irrigation ======
TimerHandle_t irrigationTimer = NULL;

// ====== Shared Data Structure ======
typedef struct {
  float temperature;
  float humidity;
  int soil;
  int mq;
} SensorData;

typedef struct {
  int triggerType;
  unsigned long time;
} IrrigationCommand;

// ====== Blynk Button State ======
bool blynkManual = false;

// ====== Circular Buffer for Sensor Data ======
#define BUFFER_SIZE 5
SensorData sensorBuffer[BUFFER_SIZE];
int bufferIndex = 0;

// ====== BLE Server Callbacks ======
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    bleDeviceConnected = true;
    Serial.println("BLE Client Connected!");
  }

  void onDisconnect(BLEServer* pServer) {
    bleDeviceConnected = false;
    Serial.println("BLE Client Disconnected!");
    // Auto restart advertising
    pServer->startAdvertising();
  }
};

// ====== SOFTWARE TIMER CALLBACK ======
void irrigationTimerCallback(TimerHandle_t xTimer) {
```

```
  IrrigationCommand irrCmd;
  irrCmd.triggerType = 1;
  irrCmd.time = millis();

  if (xQueueSend(irrigationQueue, &irrCmd, 0) != pdPASS) {
    Serial.println("WARNING: Irrigation queue penuh dari Timer!");
  }
}

// Blynk V0 → manual control
BLYNK_WRITE(V0) {
  if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
    blynkManual = param.asInt();
    xSemaphoreGive(blynkMutex);
  }
}

void irrigate() {
  if (xSemaphoreTake(pumpMutex, portMAX_DELAY) == pdTRUE) {
    Serial.println(">>> AIR AIR AIR <<<");
    digitalWrite(PUMP_LED, HIGH);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    digitalWrite(PUMP_LED, LOW);
    Serial.println(">>> IRIGASI SELESAI <<<");
    xSemaphoreGive(pumpMutex);
  }
}

int readSoil() {
  int raw = analogRead(SOIL_PIN);
  int moisture = map(raw, 4095, 1500, 0, 100);
  return constrain(moisture, 0, 100);
}

int readMQ() {
  int mq = analogRead(MQ_PIN);
  return mq;
}

// ====== TASK FUNCTIONS ======
void vSensorTask(void *pvParameter) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xPeriod = pdMS_TO_TICKS(1000);

  while (1) {
    SensorData sensorData;

    if (xSemaphoreTake(sensorMutex, portMAX_DELAY) == pdTRUE) {
      sensorData.temperature = dht.readTemperature();
      sensorData.humidity = dht.readHumidity();
      xSemaphoreGive(sensorMutex);
    }
```

```
    sensorData.soil = readSoil();
    sensorData.mq = readMQ();

    if (xQueueSend(sensorQueue, &sensorData, pdMS_TO_TICKS(100)) !=
pdPASS) {
      Serial.println("WARNING: Sensor queue penuh!");
    }

    if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
      sensorBuffer[bufferIndex] = sensorData;
      bufferIndex = (bufferIndex + 1) % BUFFER_SIZE;
      xSemaphoreGive(blynkMutex);
    }

    vTaskDelayUntil(&xLastWakeTime, xPeriod);
  }
}

void vControlTask(void *pvParameter) {
  SensorData sensorData;
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xPeriod = pdMS_TO_TICKS(1000);

  while (1) {
    if (xQueueReceive(sensorQueue, &sensorData, 0) == pdTRUE) {
      Serial.println("\n========== MONITOR ==========");
      Serial.print("Suhu: "); Serial.println(sensorData.temperature);
      Serial.print("Kelembapan Udara: ");
Serial.println(sensorData.humidity);
      Serial.print("Soil Moisture: ");
Serial.println(sensorData.soil);
      Serial.print("Kualitas Udara: ");
Serial.println(sensorData.mq);

      // ================ WARNING SYSTEM ======================
      if (sensorData.temperature > tempThreshold) {
        Serial.println("WARNING: Suhu terlalu tinggi!");
        if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
          Blynk.logEvent("warning_temp", "Suhu di atas batas!");
          xSemaphoreGive(blynkMutex);
        }
      }

      if (sensorData.humidity < humThreshold) {
        Serial.println(" WARNING: Kelembapan udara rendah!");
        if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
          Blynk.logEvent("warning_hum", "Kelembapan udara rendah!");
          xSemaphoreGive(blynkMutex);
        }
      }

      if (sensorData.mq > mqThreshold) {
        Serial.println("WARNING: Kualitas udara buruk!");
```

```
        if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
          Blynk.logEvent("warning_mq", "Kualitas udara buruk!");
          xSemaphoreGive(blynkMutex);
        }
      }

      // ================ IRRIGATION CONTROL ==================
      if (sensorData.soil < soilThreshold) {
        IrrigationCommand irrCmd;
        irrCmd.triggerType = 2;
        irrCmd.time = millis();

        if (xQueueSend(irrigationQueue, &irrCmd, pdMS_TO_TICKS(100))
!= pdPASS) {
          Serial.println("WARNING: Irrigation queue penuh!");
        }
      }
    }

    IrrigationCommand irrCmd;
    if (xQueueReceive(irrigationQueue, &irrCmd, 0) == pdTRUE) {
      switch (irrCmd.triggerType) {
        case 1:
          Serial.println("Pemicu: SOFTWARE TIMER");
          irrigate();
          break;
        case 2:
          Serial.println("Pemicu: SOIL DI BAWAH THRESHOLD");
          irrigate();
          break;
        case 3:
          Serial.println("Pemicu: TOMBOL LOKAL");
          irrigate();
          break;
        case 4:
          Serial.println("Pemicu: BLYNK BUTTON");
          irrigate();
          break;
      }
    }

    vTaskDelayUntil(&xLastWakeTime, xPeriod);
  }
}

void vButtonTask(void *pvParameter) {
  bool lastButtonState = HIGH;
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xPeriod = pdMS_TO_TICKS(50);

  while (1) {
    bool currentButtonState = digitalRead(LOCAL_BUTTON);
```

```
    if (lastButtonState == HIGH && currentButtonState == LOW) {
      IrrigationCommand irrCmd;
      irrCmd.triggerType = 3;
      irrCmd.time = millis();

      if (xQueueSend(irrigationQueue, &irrCmd, pdMS_TO_TICKS(100)) !=
pdPASS) {
        Serial.println("WARNING: Irrigation queue penuh!");
      }
      vTaskDelay(300 / portTICK_PERIOD_MS);
    }

    lastButtonState = currentButtonState;
    vTaskDelayUntil(&xLastWakeTime, xPeriod);
  }
}

void vBlynkTask(void *pvParameter) {
  bool localBlynkManual = false;
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xPeriod = pdMS_TO_TICKS(2000);

  while (1) {
    if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
      if (blynkManual) {
        localBlynkManual = true;
        blynkManual = false;
      }
      xSemaphoreGive(blynkMutex);
    }

    if (localBlynkManual) {
      localBlynkManual = false;

      IrrigationCommand irrCmd;
      irrCmd.triggerType = 4;
      irrCmd.time = millis();

      if (xQueueSend(irrigationQueue, &irrCmd, pdMS_TO_TICKS(100)) !=
pdPASS) {
        Serial.println("WARNING: Irrigation queue full!");
      }

      if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
        Blynk.virtualWrite(V0, 0);
        xSemaphoreGive(blynkMutex);
      }
    }

    if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
      int latestIndex = (bufferIndex == 0) ? BUFFER_SIZE - 1 :
bufferIndex - 1;
      Blynk.virtualWrite(V1, sensorBuffer[latestIndex].temperature);
```

```
      Blynk.virtualWrite(V2, sensorBuffer[latestIndex].humidity);
      Blynk.virtualWrite(V3, sensorBuffer[latestIndex].soil);
      Blynk.virtualWrite(V4, sensorBuffer[latestIndex].mq);
      xSemaphoreGive(blynkMutex);
    }

    if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
      Blynk.run();
      xSemaphoreGive(blynkMutex);
    }

    vTaskDelayUntil(&xLastWakeTime, xPeriod);
  }
}

// ====== BLE TASK ======
void vBLETask(void *pvParameter) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xPeriod = pdMS_TO_TICKS(2000);  // Update setiap 2
detik

  while (1) {
    // Kirim data sensor via BLE jika ada client yang connect
    if (bleDeviceConnected && xSemaphoreTake(bleMutex, portMAX_DELAY)
== pdTRUE) {
      // Ambil data terbaru dari buffer
      if (xSemaphoreTake(blynkMutex, portMAX_DELAY) == pdTRUE) {
        int latestIndex = (bufferIndex == 0) ? BUFFER_SIZE - 1 :
bufferIndex - 1;
        SensorData data = sensorBuffer[latestIndex];
        xSemaphoreGive(blynkMutex);

        // Update BLE characteristics
        String tempStr = String(data.temperature, 1) + " C";
        String humStr = String(data.humidity, 1) + " %";
        String soilStr = String(data.soil) + " %";
        String mqStr = String(data.mq);

        pCharTemp->setValue(tempStr.c_str());
        pCharTemp->notify();

        pCharHum->setValue(humStr.c_str());
        pCharHum->notify();

        pCharSoil->setValue(soilStr.c_str());
        pCharSoil->notify();

        pCharMQ->setValue(mqStr.c_str());
        pCharMQ->notify();

        Serial.println("BLE: Data sent to client");
      }
      xSemaphoreGive(bleMutex);
```

```
    }

    vTaskDelayUntil(&xLastWakeTime, xPeriod);
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  pinMode(PUMP_LED, OUTPUT);
  digitalWrite(PUMP_LED, LOW);

  pinMode(LOCAL_BUTTON, INPUT_PULLUP);

  dht.begin();

  // --- WIFI ---
  Serial.print("WiFi, Wifi, ");
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("\n WiFi Connected!");
  Serial.println(WiFi.localIP());

  // --- BLYNK ---
  Serial.print("Menghubungkan ke Blynk");
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  Serial.println("\n Blynk Connected!");

  // ====== INITIALIZE BLE ======
  Serial.println("Initializing BLE...");
  BLEDevice::init("SmartFarm_K13");  // Nama BLE device

  // Create BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Create BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // Create BLE Characteristics
  pCharTemp = pService->createCharacteristic(
    CHAR_TEMP_UUID,
    BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
  );
  pCharTemp->addDescriptor(new BLE2902());

  pCharHum = pService->createCharacteristic(
    CHAR_HUM_UUID,
```

```
    BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
  );
  pCharHum->addDescriptor(new BLE2902());

  pCharSoil = pService->createCharacteristic(
    CHAR_SOIL_UUID,
    BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
  );
  pCharSoil->addDescriptor(new BLE2902());

  pCharMQ = pService->createCharacteristic(
    CHAR_MQ_UUID,
    BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
  );
  pCharMQ->addDescriptor(new BLE2902());

  // Start service
  pService->start();

  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();

  Serial.println("BLE Ready! Device name: SmartFarm_K13");

  // ====== INITIALIZE RTOS COMPONENTS ======
  sensorQueue = xQueueCreate(10, sizeof(SensorData));
  irrigationQueue = xQueueCreate(10, sizeof(IrrigationCommand));

  pumpMutex = xSemaphoreCreateMutex();
  sensorMutex = xSemaphoreCreateMutex();
  blynkMutex = xSemaphoreCreateMutex();
  bleMutex = xSemaphoreCreateMutex();

  for (int i = 0; i < BUFFER_SIZE; i++) {
    sensorBuffer[i].temperature = 0.0;
    sensorBuffer[i].humidity = 0.0;
    sensorBuffer[i].soil = 0;
    sensorBuffer[i].mq = 0;
  }

  irrigationTimer = xTimerCreate(
    "IrrigationTimer",
    pdMS_TO_TICKS(interval),
    pdTRUE,
    (void *)0,
```

```
    irrigationTimerCallback
  );

  if (irrigationTimer != NULL) {
    xTimerStart(irrigationTimer, 0);
    Serial.println("Software Timer Started!");
  }

  // Create tasks
  xTaskCreatePinnedToCore(vSensorTask, "SensorTask", 4096, NULL, 2,
&sensorTaskHandle, 0);
  xTaskCreatePinnedToCore(vControlTask, "ControlTask", 4096, NULL, 3,
&controlTaskHandle, 0);
  xTaskCreatePinnedToCore(vButtonTask, "ButtonTask", 2048, NULL, 1,
&buttonTaskHandle, 1);
  xTaskCreatePinnedToCore(vBlynkTask, "BlynkTask", 4096, NULL, 2,
&blynkTaskHandle, 1);
  xTaskCreatePinnedToCore(vBLETask, "BLETask", 4096, NULL, 2,
&bleTaskHandle, 0);

  Serial.println("RTOS Tasks Initialized!");
}

void loop() {
  vTaskDelay(portMAX_DELAY);
}
```

Appendix C: Documentation