Penyiram Tanaman Otomatis Berbasis ESP32 dengan Implementasi *Deep Sleep* dan IoT

Irfan Arif Maulana

Teknik Elektro, Fakultas Teknik, Universitas Indonesia
Depok, Indonesia

Bandung, 1 Januari 2022

ABSTRAK

Salah satu kebutuhan dalam melakukan pemeliharaan tanaman adalah penyiraman secara berkala. Dalam mengikuti trend teknologi, tidak ada salahnya untuk melakukan optimasi pada kegiatan penyiraman tanaman tersebut. Makalah ini menggunakan menggunakan ESP32 sebagai basis sistem untuk memberdayakan nilai yang terbaca oleh sensor kelembaban tanah. Berdasarkan nilai tersebut, dideterminasi apakah pompa air digerakkan atau tidak untuk menyiram tanaman. Makalah ini juga mengimplementasikan lebih lanjut fitur ESP32 seperti deep sleep untuk menghemat daya yang digunakan dan external API seperti IoT untuk melakukan akuisisi data.

Kata Kunci: ESP32, kelembaban tanah, sensor, aktuator, deep sleep, IoT.

I. PERMASALAHAN

Dalam melakukan pemeliharaan tanaman, diperlukan kegiatan penyiraman tanaman secara berkala dengan jumlah air tertentu bergantung pada tingkat kebasahan atau kelembaban tanah pada tanaman tersebut. Penyiraman tanaman yang berkala ini dapat diotomasi dengan merancang sistem berbasis ESP32. Di sini ESP32 berperan sebagai pusat kendali sistem yang akan melakukan keputusan apakah tanaman mesti disiram atau tidak. ESP32 dihubungkan oleh input berupa sensor kelembaban tanah dan menghasilkan output berupa logika yang diberikan pada sebuah *relay* yang kemudian digunakan untuk menggerakkan pompa air ke tanaman. Nilai kelembaban tanah yang didapat oleh sensor diperhitungkan pada ESP32 dan berdasarkan nilai tersebut, keputusan menggerakkan pompa air bisa diambil.

Salah satu permasalahan lain adalah penggunaan daya yang digunakan untuk membangkitkan sistem ESP32. Untuk melakukan penghematan daya yang digunakan ini, diimplementasikan fitur yang juga datang dari ESP32 itu sendiri, yaitu *deep sleep*. Selain itu, untuk melakukan pengamatan sistem secara jarak jauh, diimplementasikan pula modul WiFi untuk menghubungkan sistem ke *external* API melalui internet. Dengan begitu data-data sistem dapat diamati dari mana saja.

II. SOLUSI PERMASALAHAN

Pada sistem yang dibuat, diimplementasikan beberapa teori dan fitur yang tersedia pada ESP32 bersamaan dengan perangkat-perangkat eksternal yang mendukung. Teori yang diimplementasikan ini antara lain adalah:

1. ***External Interfacing*** **(Input dan Output)**

   Sistem menerima input dari sensor kelembaban tanah yang akan mengukur tingkat basah atau lembabnya tanah pada pot tanaman. Sensor kelembaban tanah yang digunakan pada makalah ini bernomor seri YL-69. Sebelum terhubung dengan ESP32, sensor ini dihubungkan terlebih dahulu dengan potensiometer untuk dapat divariasikan sensitivitas sensor dalam pembacaan kelembaban tanah. Data analog yang diperoleh dari sensor ini kemudian diberikan ke ESP32 untuk diolah dan dijadikan acuan untuk keputusan output sistem. Output sistem ini berupa logika yang diberikan pada sebuah *relay* HW-307 yang bekerja secara *active* LOW. Dalam kata lain, diperlukan logika LOW yang diberikan untuk mengaktifkan *relay*. *Relay* ini terhubung dengan pompa air untuk menyiram tanaman. Oleh karena diinginkan secara *default* pompa tidak bergerak pada saat *relay* dalam keadaan mati, maka saluran listrik yang terhubung ke pompa disambung ke bagian *normally open* pada modul *relay*. Kemudian untuk kemudahan *wiring* sistem, digunakan *breadboard* 400 titik.

2. ***Web Server*** **dan** ***External*** **API**

   Sistem yang dirancang ditujukan mampu untuk dapat dilakukan pengamatan jarak jauh. Oleh karena itu, dibutuhkan sebuah *external* API yang dapat menampilkan data-data dari sistem yang dapat diakses melalui internet. Supaya sistem dapat terhubung ke internet, diimplementasikanlah modul WiFi yang tersedia dari ESP32 itu sendiri. Modul WiFi ini memungkinkan dan membukakan kemampuan ESP32 untuk berkomunikasi melalui internet dan mendapat segala macam fitur lainnya. Selain penggunaan *platform* IoT berupa Thinger.io sebagai *external* API-nya, sistem ini juga mengambil data waktu dari server menggunakan *library* NTPClient. Tujuan mengambil data waktu ini adalah untuk mengukur lama waktu sistem melakukan *deep sleep* dan menjadwalkan kapan sistem akan melakukan perhitungan kelembaban tanah dan mendorong pompa.

3. ***Deep Sleep*** **dan** ***Watch Dog***

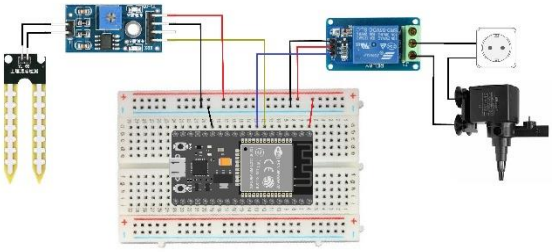
   Dalam melakukan penghematan daya yang digunakan untuk membangkitkan sistem ini, diterapkan kemampuan ESP32 yaitu *deep sleep*. Dengan begitu, sistem ini dapat bekerja untuk jangka waktu yang panjang tanpa harus repot mengganti baterai yang digunakan dalam waktu yang berdekatan. Selain itu, diterapkan pula *watch dog timer* yang difungsikan untuk melakukan *interrupt* manakala WiFi tidak berhasil dihubungkan sampai pada jangka waktu tertentu.

III. METODOLOGI

Perangkat-perangkat yang digunakan pada sistem ini antara lain adalah:

1. ESP32
2. Sensor kelembaban tanah YL-69
3. Potensiometer
4. *Relay* HW-307
5. Pompa air aquarium
6. *Breadboard*
7. Kabel listrik dan stop kontak
8. Kabel *jumper*

Perangkat-perangkat tersebut dirangkai mengikuti skema berikut:



Gambar 1. Skema Rangkaian Sistem

Ada pun hubungan pin-pin antar perangkat eksternal adalah sebagai berikut:

Komponen	**Pin Komponen**	**Pin ESP32**
Sensor YL-69	VCC	(ke potensio)
	GND	
Potensiometer	VCC	3V3
	GND	GND
	AO	32
Relay HW-307	VCC	3V3
	GND	GND
	IN	33

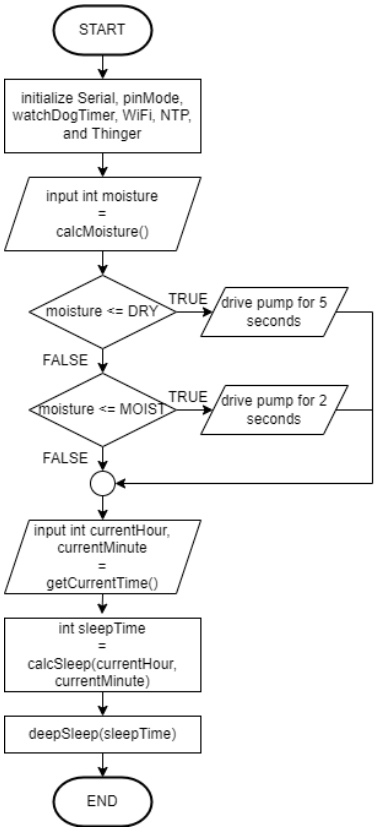
Tabel 1. Hubungan Pin

Perlu diketahui bahwa sensor YL-09 tidak langsung dihubungkan ke ESP32. Melainkan dihubungkan terlebih dahulu ke potensiometer. Potensiometer ini mampu untuk mengalibrasi sensitivitas pengukuran kelembaban tanah oleh sensor melalui sinyal yang dikirim ke ESP32. Sinyal yang dikirimkan dari potensiometer ke ESP32 ini adalah sinyal analog.

Pompa air dihubungkan ke listrik dari rumah dengan satu titik pada kabel tersebut diputuskan dan dihubungkan ke *relay* yang berperan sebagai *switch*. Oleh karena diinginkan secara *default* aliran listrik ke pompa terputus atau terbuka, maka koneksi yang dihubungkan ke *relay* merupakan pada *normally open*. Sedangkan jika *relay* diberi nilai logika tertentu, maka *relay* akan aktif dan mengubah aliran listrik menjadi tertutup.

Sensor kelembaban tanah menunjukkan tingkat kelembaban tanah dengan meneruskan besar tegangan ke ESP32 yang diambil berupa nilai analog. Di mana semakin kering tanah yang terukur, semakin tinggi nilai tegangan yang diberikan. Atau dalam kata lain semakin besar nilai analog yang diteruskan ke ESP32. Ada pun pada makalah ini, dirancang bahwa semakin kecil nilai *moisture* yang ada pada program, semakin kering tanah tersebut. Serta nilai *moisture* yang diukur ini berada pada skala 0 hingga 100. Dengan demikian, diberikan fungsi *map()* yang akan memetakan nilai analog yang terukur dari 0 hingga 4095 menjadi 100 hingga 0. Dari skala 100 ini, ditetapkan pula dua buah parameter, yaitu DRY dan MOIST yang masing-masing secara berurutan bernilai 25 dan 50 yang menunjukkan apakah tanah kering atau lembab. Kedua nilai ini akan menentukan apakah pompa mesti digerakkan atau tidak dan seberapa lama pompa digerakkan untuk menyesuaikan berapa banyak air yang diberikan.

Penerapan fitur *deep sleep* pada sistem ini sendiri menggunakan *library* NTPClient sebagai pendukung. Di mana *library* tersebut dapat mengambil data waktu dari *web server*. Waktu ini kemudian dijadikan acuan perhitungan untuk seberapa lama ESP32 melakukan *deep sleep*. Sistem ini dirancang untuk beroperasi setiap jam 6 pagi, 1 siang, dan 8 malam. ESP32 akan mengukur kelembaban tanah dan menggerakkan pompa terlebih dahulu sebelum melakukan *deep sleep* setiap kali sistem dihidupkan. Pada program, implementasi ini semua dapat dilihat melalui *block diagram* berikut:



Gambar 2. *Block Diagram*

Perangkat-perangkat sistem yang sudah berhasil dirangkai dengan benar akan terlihat seperti berikut:

Gambar 3. Arsitektur Sistem

Gambar tersebut diambil pada saat demonstrasi. Oleh karena itu, untuk kepentingan demonstrasi, digunakan dua buah ember untuk menampung air. Di mana ember pertama untuk menyimpan persediaan air dan yang kedua untuk menampung sementara air yang dialirkan dari pompa. Sebab tanaman sudah disiram dan tidak perlu disiram lagi.

IV. IMPLEMENTASI

Ketika sistem pertama kali dinyalakan, ESP32 akan mencoba menghubungkan koneksinya ke WiFi yang telah didefinisikan pada program. Jika setelah jangka waktu tertentu, atau pada makalah ini 10 detik, WiFi belum juga terhubung, maka *watch dog timer* akan melakukan *interrupt* pada sistem untuk memanggil *system reboot* pada ESP32. Keadaan ketika WiFi tidak terhubung terlihat pada *serial monitor* seperti berikut:

```
Connecting to I believe Wi can Fi
. . . . . .
Connection failed
Rebooting
Guru Meditation Error: Core  0 panic'ed (Interrupt wdt timeout on CPU0)
Core 0 register dump:
PC      : 0x4008b106  PS      : 0x00060434  A0      : 0x8008c261  A1      : 0x3ffb5630
A2      : 0x3ffbf348  A3      : 0x0000cdcd  A4      : 0xb33fffff  A5      : 0x00000001
A6      : 0x00060223  A7      : 0x0000abab  A8      : 0x0000abab  A9      : 0x3ffb5740
A10     : 0x3ffafed4  A11     : 0x400e19b4  A12     : 0x3ffbdf94  A13     : 0x400ebbb0
A14     : 0x000000f5  A15     : 0x00006590  SAR     : 0x00000018  EXCCAUSE: 0x00000005
EXCVADDR: 0x00000000  LBEG    : 0x40[illegible]0c2e0  LEND    : 0x4000c216  LCOUNT  : 0xffffffff

ELF file SHA256: 0000000000000000

Backtrace: 0x4008b106:0x3ffb5630 0x4008c25e:0x3ffb5660 0x40089ea3:0x3ffb5680 0x40089[illegible]

Core 1 register dump:
PC      : 0x4008ca0c  PS      [illegible]
```

Gambar 4. Kondisi *Interrupt* oleh *Watch Dog Timer*

Ada pun ketika WiFi sukses terhubung dengan ESP32, ESP32 akan langsung mengukur kelembaban tanah dan nilai analog yang diperoleh dipetakan ke skala 0 hingga 100 yang menyatakan kering ke lembab. Pada Gambar 5, diperlihatkan *serial monitor* untuk demonstrasi sistem yang sensornya sedang tidak dimasukkan ke dalam tanah. Atau dalam kata lain, sensor akan mengukur tanah sekering-keringnya dan meneruskan nilai *moisture* rendah ke ESP32.

```
[illegible]Connecting to I believe Wi can Fi
. . . . . .
Connected
IP address: 192.168.43.74

Moisture level is: 1/100
Soil is dry and needs water
Driving pump . . . Done

Currently, it's 16:40:53
Next operating hour is at 20:00
Going to sleep for 3 hour(s) and 19 minute(s)
==================================================
```

Gambar 5. Kondisi WiFi Terhubung dan Mengukur Kelembaban serta Waktu *Deep Sleep*

Seperti yang dilihat bahwa berdasarkan nilai kelembaban tanah (*moisture*) yang didapat pada Gambar 5, tanah dengan level yang kering tersebut mengakibatkan pada ESP32 untuk menyalakan *relay* dan menggerakkan pompa air. *Serial monitor* berhasil menunjukkan demikian. Setelah itu, *library* NTPClient akan membantu ESP32 dalam mengambil data jam dan menit untuk dihitung berapa lama ESP32 akan melakukan *deep sleep*. Seperti yang terlihat bahwa pada demonstrasi tersebut, jam menunjukkan pukul 16:40. Sedangkan operasi selanjutnya adalah jam 20:00. Dengan demikian diprogramkan bahwa ESP32 akan masuk ke kondisi *deep sleep* selama 3 jam dan 19 menit.

Gambar 6. Koneksi Perangkat pada Thinger.io

Gambar 6 menunjukkan implementasi *external* API. Pada *platform* yang digunakan, yaitu Thinger.io, terlihat bahwa perangkat ESP32 yang sudah dirancang berhasil terhubung dengan server Thinger.io ketika ESP32 sudah terhubung dengan WiFi.

Gambar 7. Monitor pada Thinger.io

Pada API Thinger.io, diperlihatkan monitor grafik per satuan waktu yang menunjukkan dua buah variabel yang berbeda. Yaitu, variabel kelembaban tanah itu sendiri atau *moisture* dan variabel yang menunjukkan apakah pompa air digerakkan atau tidak (*pump*). Seperti yang diketahui bahwa *moisture* berskala 0 hingga 100 dengan keterangan tanah kering hingga basah. Sedangkan *pump* menyatakan nilai Boolean nyala atau matinya pompa. Dengan nol berarti mati dan satu berarti menyala. Pada demonstrasi yang diperlihatkan pada Gambar 7, ESP32 sudah dimatikan kemampuannya untuk melakukan *deep sleep* agar interval pengambil data menjadi lebih singkat dan mudah untuk diamati. Gambar 7 menunjukkan bahwa terdapat titik-titik yang ada pada kedua grafik nilai variabel tersebut. Ini menandakan komunikasi antar ESP32 dengan *external* API berhasil dilakukan dengan pembacaan nilai variabel yang tepat. Dalam hal ini, sensor sedang tidak dimasukkan ke dalam tanah sehingga nilainya konstan satu baik untuk *moisture* maupun *pump*-nya.

V. DISKUSI PENYELESAIAN MASALAH

Dilihat dari fungsi utamanya, sistem ini sudah bekerja dengan semestinya. Sistem berhasil menerima input dan menghasilkan output yang sesuai dengan yang dirancang.

Hal utama yang menjadi permasalahan sistem ini adalah komunikasi untuk memberikan data ke *external* API yang digunakan. Di mana *platform* IoT yang digunakan, yaitu Thinger.io, membutuhkan waktu yang cukup lama untuk merespon dengan ESP32 dan menerima komunikasi data. Terdapat cukup lama waktu yang dibutuhkan agar data dapat ditampilkan ke layer *widget* pada Thinger.io karena respon server yang kurang mendukung. Pada waktu singkat siklus operasional ESP32 yang dirancang, data menjadi tidak sempat diterima oleh Thinger.io. Untuk mengatasi masalah ini, diberikan sebuah fungsi *delay()* yang mengantisipasi lamanya waktu koneksi ESP32 dengan server Thinger.io.

Lampiran *Source Code*

```
// ===== Libraries =====
#include <WiFi.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <ThingerESP32.h>

// ===== Pin Assignments =====
#define sensorPin 32
#define relayPin  33

// ===== Moisture and Pump Variable and Parameters =====
int moisture, pump = 0;
#define DRY   25
#define MOIST 50

// ===== Timer Parameters =====
#define S_TO_uS 1000000
#define M_TO_S  60
#define H_TO_S  3600

// ===== WatchDog Variables =====
#define WATCHDOG_TIMEOUT_S 10
hw_timer_t * watchDogTimer = NULL;

// ===== WiFi Connections and NTP Client =====
const char* WIFI_SSID = "YOUR_SSID";
const char* WIFI_PASS = "YOUR_PASSWORD";
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

// ===== Thinger =====
#define USERNAME          "YOUR_USERNAME"
#define DEVICE_ID         "YOUR_DEVICE"
#define DEVICE_CREDENTIAL "YOUR_CREDENTIAL"
ThingerESP32 thing(USERNAME, DEVICE_ID, DEVICE_CREDENTIAL);

// ===== Timer Variables =====
String formattedTime;
int currentHour, currentMinute, sleepHour = 0, sleepMinute = 0;
unsigned long sleepTime = 0;

// ===== Modular Functions Declaration =====
void IRAM_ATTR watchDogInterrupt();
void watchDogRefresh();
void watchDogInit();

void connectWiFi();
void NTPClientInit();

int calcMoisture();
void drivePump();

void getCurrentTime();
void sleep();
```

```
// ===== Setup =====
void setup() {
  delay(5000);
  Serial.begin(9600);
  pinMode(sensorPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);

  connectWiFi();
  NTPClientInit();

  thing["moisture"] >> outputValue(moisture);
  thing["pump"] >> outputValue(pump);
}

// ===== Loop =====
void loop() {
  thing.handle();
  moisture = calcMoisture();
  drivePump();
  sleep();
  delay(5000);
}
```

```
// ===== Modular Functions Definition =====
void IRAM_ATTR watchDogInterrupt() {
  Serial.println("Rebooting");
  delay(2000);
  ESP.restart();
}

void watchDogRefresh() {
  timerWrite(watchDogTimer, 0);
}

void watchDogInit() {
  watchDogTimer = timerBegin(0, 80, true);
  timerAttachInterrupt(watchDogTimer, &watchDogInterrupt, true);
  timerAlarmWrite(watchDogTimer, WATCHDOG_TIMEOUT_S * S_TO_uS, false);
  timerAlarmEnable(watchDogTimer);
}

void NTPClientInit() {
  timeClient.begin();
  timeClient.setTimeOffset(7 * 3600); // GMT + 7
}

int calcMoisture() {
  return map(analogRead(sensorPin), 0, 4095, 100, 0);
}

void getCurrentTime() {
  timeClient.update();
  formattedTime = timeClient.getFormattedTime();
  currentHour = timeClient.getHours();
  currentMinute = timeClient.getMinutes();
}
```

```
void connectWiFi() {
  long startTime, loopTime;

  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.println(WIFI_SSID);

  watchDogInit();
  watchDogRefresh();

  startTime = millis();
  while(WiFi.status() != WL_CONNECTED) {
    Serial.print(". ");
    delay(500);
    loopTime = millis() - startTime;

    if(loopTime > 9000) {
      Serial.println("");
      Serial.println("Connection failed");
      delay(1000);
    }
  }

  timerAlarmDisable(watchDogTimer);
  Serial.println("");
  Serial.println("Connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("");
}
```

```
void drivePump() {
  Serial.print("Moisture level is: ");
  Serial.print(moisture);
  Serial.println("/100");

  if(moisture <= DRY) {
    Serial.println("Soil is dry and needs water");
    Serial.print("Driving pump . . . ");
    digitalWrite(relayPin, LOW);
    pump = 1;
    delay(5000);
    digitalWrite(relayPin, HIGH);
    Serial.println("Done");
  } else if(moisture > DRY && moisture <= MOIST) {
    Serial.println("Soil is quite moist and needs some water");
    Serial.print("Driving pump . . . ");
    digitalWrite(relayPin, LOW);
    pump = 1;
    delay(2000);
    digitalWrite(relayPin, HIGH);
    Serial.println("Done");
  } else {
    Serial.println("Soil is already wet");
    Serial.println("No need to drive pump");
    digitalWrite(relayPin, HIGH);
    pump = 0;
  }
  Serial.println("");
}
```

```
void sleep() {
  getCurrentTime();

  Serial.print("Currently, it's ");
  Serial.println(formattedTime);

  if(currentHour >= 6 && currentHour < 13) {
    sleepHour = 13 - currentHour;
    Serial.println("Next operating hour is at 13:00");
  } else if(currentHour >= 13 && currentHour < 20) {
    sleepHour = 20 - currentHour;
    Serial.println("Next operating hour is at 20:00");
  } else if(currentHour >= 20 && currentHour < 24) {
    sleepHour = 24 - currentHour + 6;
    Serial.println("Next operating hour is at 6:00");
  } else if(currentHour < 6) {
    sleepHour = 6 - currentHour;
    Serial.println("Next operating hour is at 6:00");
  }

  if(currentMinute > 0) {
      sleepHour -= 1;
      sleepMinute = 59 - currentMinute;
  }

  sleepTime = sleepHour * H_TO_S;
  sleepTime += sleepMinute * M_TO_S;
  sleepTime = sleepTime * S_TO_uS;

  if(sleepTime > 0) {
    Serial.print("Going to sleep for ");
    Serial.print(sleepHour);
    Serial.print(" hour(s) and ");
    Serial.print(sleepMinute);
    Serial.println(" minute(s)");
    Serial.println("=================================================");
    esp_sleep_enable_timer_wakeup(sleepTime);
    Serial.flush();
    delay(3000);
    esp_deep_sleep_start();
  }
}
```