Phase 1: Simulation & Prototyping (in Wokwi)

1.1 Objective

The primary objective of this initial phase was to **validate the project's core logic and software in a low-cost, risk-free environment.**

Before committing to designing and paying for a physical Printed Circuit Board (PCB),which is permanent and difficult to change, it was essential to prove that the code could work as intended. This "software-first" approach de-risks the entire project by allowing us to find and fix any bugs in the logic *before* building the hardware. The goal was to confirm that the ESP32 could:

1. Correctly read data from both the digital DHT22 sensor and the analog Soil Moisture Sensor.
2. Process the raw sensor data into a human-readable format (percentage and degrees Celsius).
3. Display this information clearly on an I2C LCD screen.
4. Execute a control logic if statement to make a decision.
5. Successfully control an output (the LED) based on that decision.

1.2. Tools & Components

The simulation was conducted using the **Wokwi online electronics platform**. The virtual components selected for this prototype (as seen in the diagram) were:

- **Controller:** ESP32-DevKitC (The "brain" of the project).
- **Temperature/Humidity Sensor:** DHT22 (A digital sensor to measure ambient conditions).
- **Soil Moisture Sensor:** Capacitive Soil Moisture Sensor (An analog sensor that provides a variable voltage based on how wet the soil is).
- **Display:** 16x2 I2C LCD (A simple screen that only requires two pins, SDA and SCL, for communication).
- **Indicator (Pump Simulator):** 1x Red LED (Acts as a visual stand-in for the real-world water pump).
- **Support Component:** 1x Resistor (Used as a current-limiting resistor to protect the LED from burning out).

1.3. Methodology

The system was designed to run in a continuous loop, exactly as it would in the real world. The code (provided in Appendix A) executes the following steps:

1. **Initialization (void setup()):**
    - The ESP32 initializes the Serial monitor (for debugging), the DHT22 sensor, and the I2C LCD screen.
    - It defines PUMP_LED_PIN (GPIO13) as an OUTPUT and sets it to LOW (off) by default.
    - It prints a "Smart Irrigation" welcome message to the LCD.
2. **Continuous Loop (void loop()):**
    - **Read Sensors:** The ESP32 "polls" the sensors for new data. It calls dht.readTemperature() to get the temperature from GPIO15 and analogRead(SOIL_MOISTURE_PIN) to get a raw value from GPIO34.
    - **Process Data:** The raw soil moisture value is a 12-bit number from 0 to 4095. This is not user-friendly. The code uses the map(moistureRaw, 4095, 0, 0, 100) function to convert this number into an intuitive 0-100% scale.
    - **Display Data:** The code updates the LCD screen with the latest Temperature and Soil % values.
    - **Control Logic:** This is the core of the project. An if statement checks the moisture level against a moistureThreshold (set to 40%).
        - **If moisturePercent < 40 (Soil is Dry):** The code executes digitalWrite(PUMP_LED_PIN, HIGH), sending 3.3V to the LED and turning it ON (simulating the pump).
        - **Else (Soil is Wet):** The code executes digitalWrite(PUMP_LED_PIN, LOW), turning the LED OFF.
    - **Delay:** The system waits for 5 seconds (delay(5000)) before repeating the loop.

1.4. Result

The simulation was **100% successful**. The Wokwi environment confirmed that:

- The ESP32 correctly read and processed data from both analog and digital sensors.
- The I2C communication with the LCD was stable and displayed the correct values.
- The core if/else control logic worked perfectly, turning the LED on and off at the exact 40% moisture threshold.

This successful simulation proved that the project's concept and code were sound, giving us the necessary confidence to proceed with designing a permanent, custom PCB.

CODE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

// Pin Definitions
#define DHTPIN 15          // DHT22 data pin to GPIO15
#define DHTTYPE DHT22
#define SOIL_MOISTURE_PIN 34 // Analog input pin
#define PUMP_LED_PIN 13     // LED to indicate pump status (GPIO13)

// Threshold for dry soil (in %)
int moistureThreshold = 40;

// Initialize LCD and DHT
LiquidCrystal_I2C lcd(0x27, 16, 2);
DHT dht(DHTPIN, DHTTYPE);

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

  // Initialize components
  lcd.init();
  lcd.backlight();

  dht.begin();

  pinMode(PUMP_LED_PIN, OUTPUT);
  digitalWrite(PUMP_LED_PIN, LOW); // Start with LED OFF (soil wet)

  lcd.setCursor(0, 0);
  lcd.print("Smart Irrigation");
  delay(2000);
  lcd.clear();
}

void loop() {
  // Read temperature & humidity
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Read soil moisture
  int moistureRaw = analogRead(SOIL_MOISTURE_PIN);
  int moisturePercent = map(moistureRaw, 4095, 0, 0, 100); // ESP32 uses 12-bit ADC (0-
4095)

  // Display on LCD
  lcd.setCursor(0, 0);
```

```
  lcd.print("Temp:");
  lcd.print(temperature, 1);
  lcd.print((char)223); // Degree symbol
  lcd.print("C ");

  lcd.setCursor(0, 1);
  lcd.print("Soil:");
  lcd.print(moisturePercent);
  lcd.print("%   ");

  // Debug Info
  Serial.println("================================");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" *C");

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");

  Serial.print("Soil Moisture: ");
  Serial.print(moisturePercent);
  Serial.println(" %");

  // Simulated Email Alert
  if (temperature > 35) {
    Serial.println(">> ALERT: High Temp! (Would send Email)");
  }

  // LED as Pump Indicator (ON when soil is dry)
  if (moisturePercent < moistureThreshold) {
    digitalWrite(PUMP_LED_PIN, HIGH); // Turn LED ON (soil is dry)
    Serial.println("Pump Status: ON (Soil is Dry)");
  } else {
    digitalWrite(PUMP_LED_PIN, LOW); // Turn LED OFF (soil is wet)
    Serial.println("Pump Status: OFF (Soil is Wet)");
  }

  Serial.println("================================\n");

  delay(5000); // Wait before next reading
}
```

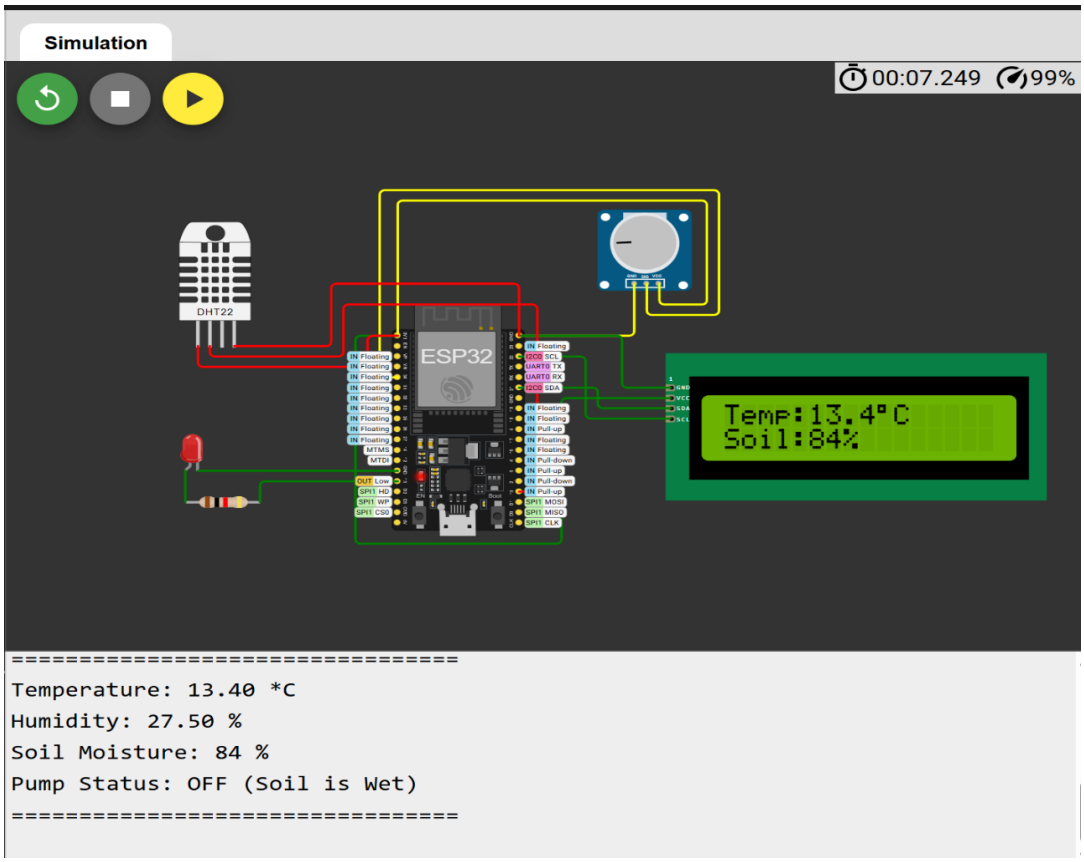


Figure_1

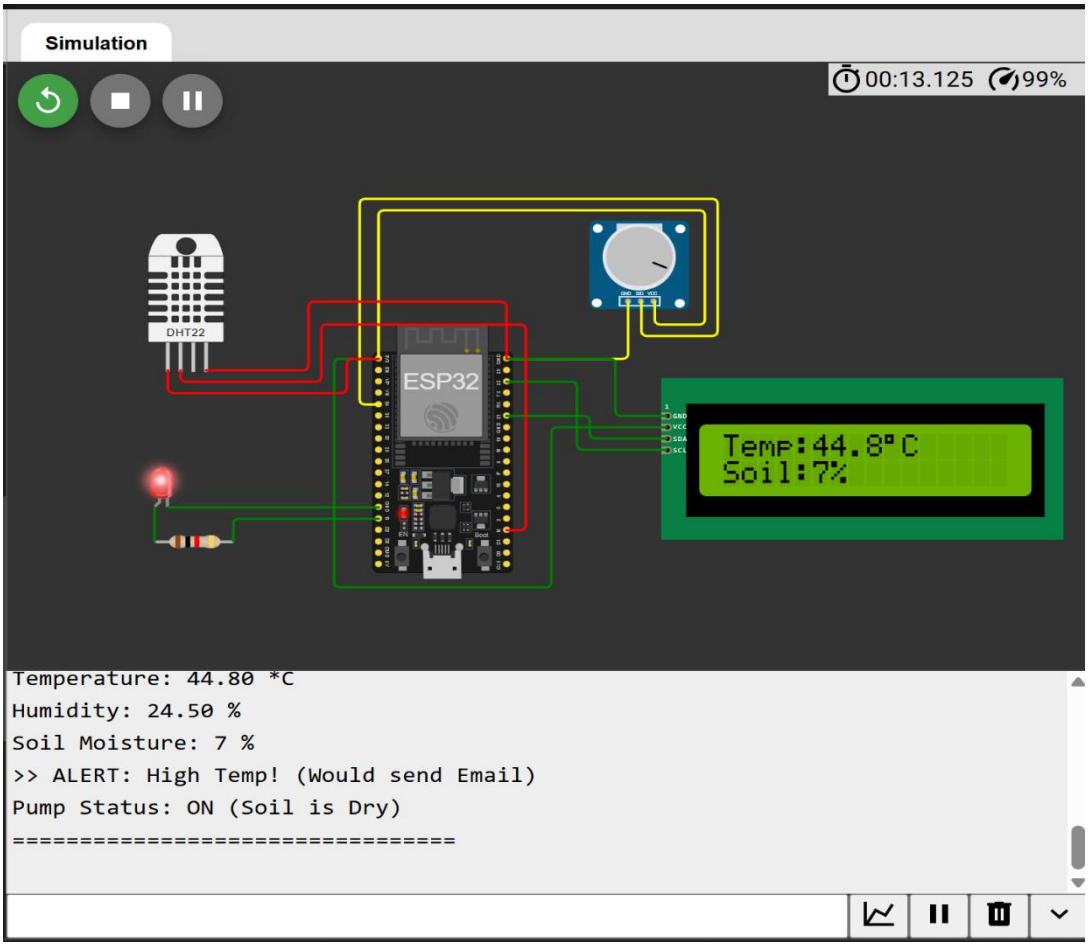


Figure_2

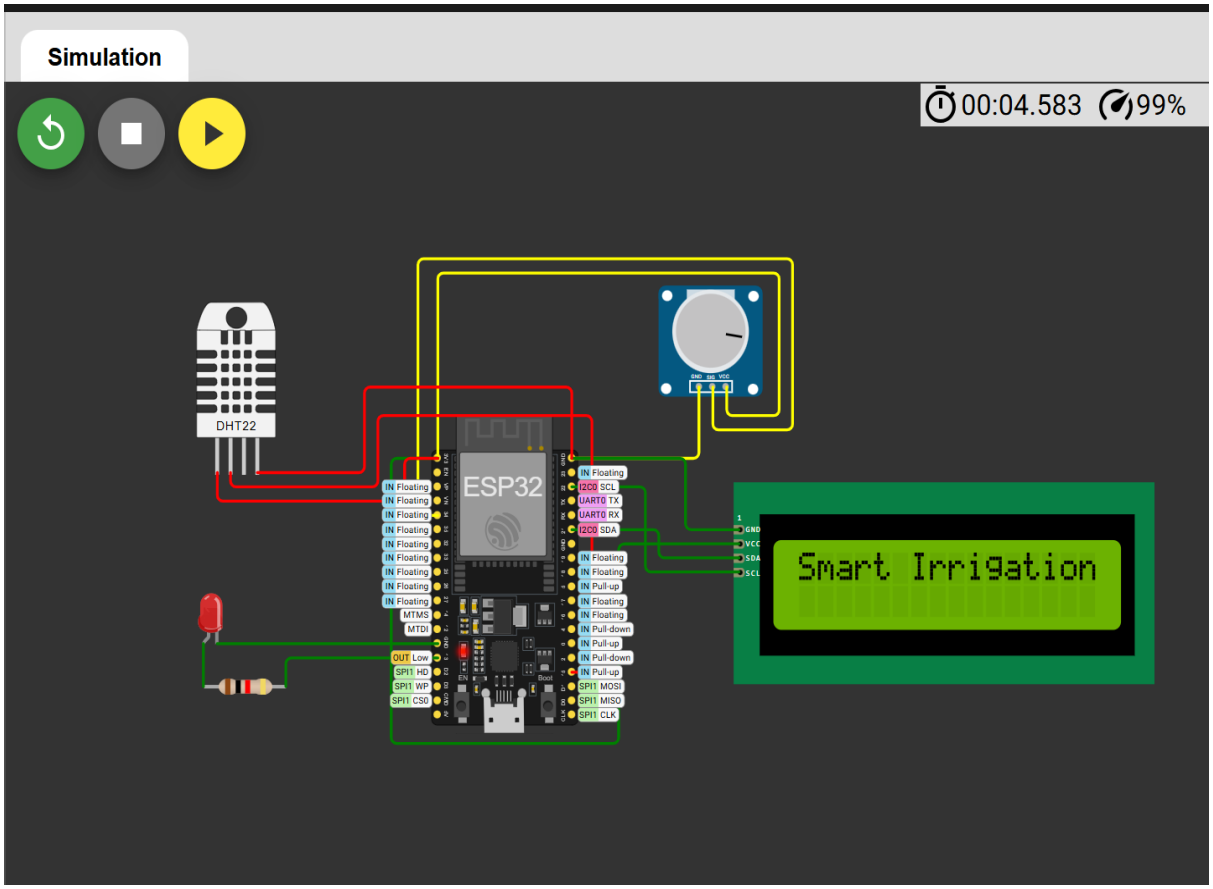


Figure_3

Phase 2: Schematic Design (The "Blueprint")

2.1. Design Goal

With the software logic proven in Wokwi, the project's next phase was to design the permanent, physical hardware. The objective was to create a formal schematic (the "blueprint") in the **EasyEDA** design suite.

A critical decision was made at this stage: instead of simply attaching a pre-made ESP32 "DevKit" to a PCB, we would design a **custom board from scratch**. This means we would use the *core* ESP32-WROOM-32D module (just the metal-can chip) and build its entire "life-support" system ourselves. This is a far more complex task, as it required us to be responsible for every aspect of the chip's operation, including:

1. **The Power System:** How to take 5 Volts from a USB plug and safely, cleanly, and reliably convert it to the 3.3 Volts the ESP32 chip needs to live.
2. **The Programming System:** How to let a standard computer "talk" to the ESP32 to upload new code.
3. **The Boot System:** How to ensure the chip wakes up and starts running our code correctly every single time power is applied.

2.2. Component Selection

A total of 26 components were required for this "from-scratch" design. All parts were selected from the **"LCSC Electronics"** library within EasyEDA. This is a critical step, as it ensures that every component we choose is a real, in-stock part from a major supplier and that its "footprint" (the physical pattern of solder pads) is 100% correct.

For all passive components (the tiny resistors and capacitors), the **0805 package size** was chosen. This is a standard Surface Mount (SMD) size that is small and professional, but still large enough to be practically hand-soldered for a prototype. (A full Bill of Materials is available in Appendix B).

2.3. Wiring Philosophy: "Net Labels" (The "Sticky Note" Method)

A schematic with 26 components can have over 100 connections. Drawing a wire for every one would create a "spider's web" that is impossible to read or debug.

To prevent this, we used a professional wiring method called **"Net Labels"**:

- **The GND Symbol:** Any pin connected to this GND (ground) symbol is considered by the software to be "teleported" and connected to all other GND symbols.
- **Net Labels (e.g., 3.3v):** This is our "sticky note" tool. We draw a short wire (a "stub") from a pin (like the output of our regulator) and place a label on it named **3.3v**. Then, any other pin that needs 3.3V (like the ESP32's power pin) also gets a stub with the *exact same label*. The software knows these are all connected.

This method keeps the schematic clean, organized into logical blocks, and makes it incredibly easy to see the *logic* of the circuit.

2.4. Schematic Circuit Blocks: A Step-by-Step Explanation

We designed the schematic by wiring these logical blocks one by one.

2.4.1. The Power System (The "Heart")

- **Problem:** The USB1 port supplies 5 Volts, but our ESP32 "brain" (U1) and "translator" (U3) both require a very clean 3.3 Volt supply.
- **Solution:** We used the **U2 (LD1117S33)** chip. This is a **linear regulator**; its only job is to take the 5V on its VIN (Voltage In) pin and output a steady 3.3V on its VOUT (Voltage Out) pin.
- **Crucial Detail (The "Filters"):** Power from a USB port is "noisy" (it has ripples and spikes). A regulator also creates its own noise. To fix this, we used **smoothing capacitors**.
    - C1 (10uF) and C4 (100nF) are the **input filters**. They are placed between the 5v net and GND, right *before* the regulator, to clean up the power.
    - C2 (10uF) and C5 (100nF) are the **output filters**. They are placed between the 3.3v net and GND, right *after* the regulator, to ensure the power that goes to our ESP32 is perfectly stable.

2.4.2. The Core Chips & Communication

- **Problem:** We needed to power our main chips and allow the USB port to talk to the "translator" chip.
- **Solution:**
    - We connected the 3V3 pin of U1 (ESP32) and the VCC pin of U3 (CH340C) to our new **3.3v** net.
    - We connected the D- and D+ pins of the USB1 port directly to the D- and D+ pins of the U3 (CH340C) chip.
    - **Crucial Detail (Decoupling):** A high-speed chip like the CH340C needs its own tiny, local "water bottle" of power for quick sips. We placed **C6 (100nF)** *right next* to the U3's VCC and GND pins. This "decoupling capacitor" is essential for the chip to operate reliably.

2.4.3. The TX/RX Crossover (The "Translator")

- **Problem:** The CH340C (translator) needs to talk to the ESP32 (brain). One chip's "mouth" (TX - Transmit) must connect to the other's "ear" (RX - Receive).
- **Solution:** We "crossed the streams" using Net Labels:
    - U3 (CH340C) TXD pin ("mouth") → ESP_RX Net Label
    - U1 (ESP32) RXD0 pin ("ear") → ESP_RX Net Label
    - (And the reverse for the other direction)
    - U3 (CH340C) RXD pin → ESP_TX Net Label
    - U1 (ESP32) TXD0 pin → ESP_TX Net Label

2.4.4. The Boot & Auto-Program Circuit (The "Ignition")

- **Problem:** The ESP32's EN (Enable) and IO0 (Boot) pins are "floating." If left unconnected, they can be randomly triggered by static, causing the chip to crash or not start. We also need a way to *automatically* put the chip into "programming mode" without having to press buttons.
- **Solution:** We built a multi-part circuit:
    - **EN (Reset) Pin:** This pin is the main "ON" switch.
        1. **Pull-up Resistor:** We connected R5 (10k) from EN to 3.3v. This "pulls the pin up" and holds it firmly in the "ON" state.
        2. **Manual Reset:** We wired SW1 (button) from EN to GND. When we press it, it pulls the pin "LOW" and resets the chip.
        3. **Noise Filter:** We added C3 (1uF) from EN to GND to absorb any tiny electrical noise spikes.
    - **IO0 (Boot) Pin:** This pin tells the chip *how* to start.

        1. **Pull-up Resistor:** We connected R4 (10k) from IO0 to 3.3v. When IO0 is "HIGH" on boot, it tells the chip: "Run your normal code."
        2. **Manual Boot:** We wired SW2 (button) from IO0 to GND. If we hold this button down during startup, it pulls the pin "LOW," which tells the chip: "Stop! Wait for new code."
    - **Auto-Program (The "Automatic Fingers"):** This is the clever part.

        1. The CH340C (translator) has two "signal" pins: DTR and RTS.
        2. We wired DTR to a transistor (Q2) which is connected to the EN pin.
        3. We wired RTS to another transistor (Q1) which is connected to the IO0 pin.
        4. When you click "Upload" in your code editor, the computer tells the CH340C to "wiggle" the DTR and RTS pins. This triggers the transistors, which automatically "press" the RESET and BOOT buttons for you in the perfect sequence.

2.4.5. Strapping Resistors (The "Secret Handshake")

- **Problem:** The ESP32 needs a "secret handshake" on boot to know *where* to load its code from (e.g., the internal flash chip).
- **Solution:** We added three 10k resistors (R1, R2, R3) as "strapping resistors." By pulling TXD0 (via R1) HIGH, and IO2 (via R2) and GPIO15 (via R3) LOW (to GND), we set the correct "boot mode."

2.4.6. The I/O (Project) Circuit

- **Problem:** Finally, we needed to connect our actual project components.
- **Solution:** This was the easy part. We just wired the headers (U4, U5, U6) and the LED1 (with its R6 "bodyguard" resistor) to the correct GPIO pins as defined in our Wokwi code (IO21, IO22, IO15, IO34, IO13).

2.5. Final Schematic Cleanup

The "Design Manager" showed warnings for all the ESP32 pins we didn't use. To create a 100% finished schematic, we used the **"No Connect Flag"** (a blue X). We placed an X on every unused pin, which tells the software, "I am *intentionally* leaving this pin unconnected." This cleared all warnings.

1. Power System

- **USB1 (Micro USB):**
  - VBUS pin → **5v** Net
  - GND pin → **GND**
- **U2 (3.3V Regulator):**
  - VIN pin → **5v** Net
  - VOUT pins (Pin 2 & 4) → **3.3v** Net
  - GND pin → **GND**
- **Power Capacitors:**
  - C1 (10uF) & C4 (100nF) → Connected between **5v** and **GND**.
  - C2 (10uF) & C5 (100nF) → Connected between **3.3v** and **GND**.

2. Core Chips & Communication

- **U1 (ESP32):**
  - 3V3 pin → **3.3v** Net
  - GND pins → **GND**
- **U3 (CH340C):**
  - VCC pin → **3.3v** Net
  - GND pin → **GND**
  - D+ pin → **D+** Net
  - D- pin → **D-** Net
- **USB Data:**
  - USB1 D+ pin → **D+** Net
  - USB1 D- pin → **D-** Net
- **Decoupling Capacitor:**
  - C6 (100nF) -> Connected between U3 VCC pin and GND.
- **TX/RX Crossover:**
  - U3 (TXD) → **ESP_RX** Net → U1 (RXD0)
  - U3 (RXD) → **ESP_TX** Net → U1 (TXD0)

3. Boot & Auto-Program Circuit

- **EN Net (ESP32 Pin 3):**
  - Connected to R5 (10k), which connects to **3.3v**. (Pull-up)
  - Connected to C3 (1uF), which connects to **GND**.
  - Connected to SW1 (Button), which connects to **GND**.
  - Connected to Collector pin of Q2 (Transistor).
- **Io0 Net (ESP32 Pin 25):**
  - Connected to R4 (10k), which connects to **3.3v**. (Pull-up)
  - Connected to SW2 (Button), which connects to **GND**.
  - Connected to Collector pin of Q1 (Transistor).

- **Auto-Program Transistors:**
  - **Q2 (Reset):** Emitter → **GND**. Base → U7 (1k) → DTR# pin of U3.
  - **Q1 (Boot):** Emitter → **GND**. Base → U8 (1k) → RTS# pin of U3.
- **Strapping Resistors:**
  - R3 (10k) →Connected between GPIO15 (Pin 23) and **GND**. (Pull-down)
  - R2 (10k) → Connected between IO2 (Pin 24) and **GND**. (Pull-down)
  - R1 (10k) → Connected between **ESP_TX** Net (Pin 35) and **3.3v**. (Pull-up)

4. Project I/O Circuit

- **U4 (LCD Header):**
  - Pin 1 → **GND**
  - Pin 2 → **3.3v** Net
  - Pin 3 → **SDA** Net (connected to IO21 on U1)
  - Pin 4 → **SCL** Net (connected to IO22 on U1)
- **U5 (DHT22 Header):**
  - Pin 1 → **3.3v** Net
  - Pin 2 → **GPIO15** Net (connected to IO15 on U1)
  - Pin 3 → **GND**
- **U6 (Soil Header):**
  - Pin 1 → **3.3v** Net
  - Pin 2 → **SOIL_AOUT** Net (connected to IO34 on U1)
  - Pin 3 → **GND**
- **LED1 (Pump LED):**
  - Cathode (-) → **GND**
  - Anode (+) → R6 (330R) →**LED_PUMP** Net (connected to IO13 on U1)

5. Unused Pins

- All other unused pins on U1 (ESP32) and the buttons (SW1, SW2) were connected to a **"No Connect" Flag** (blue X).

Schematic Sketch for PCB

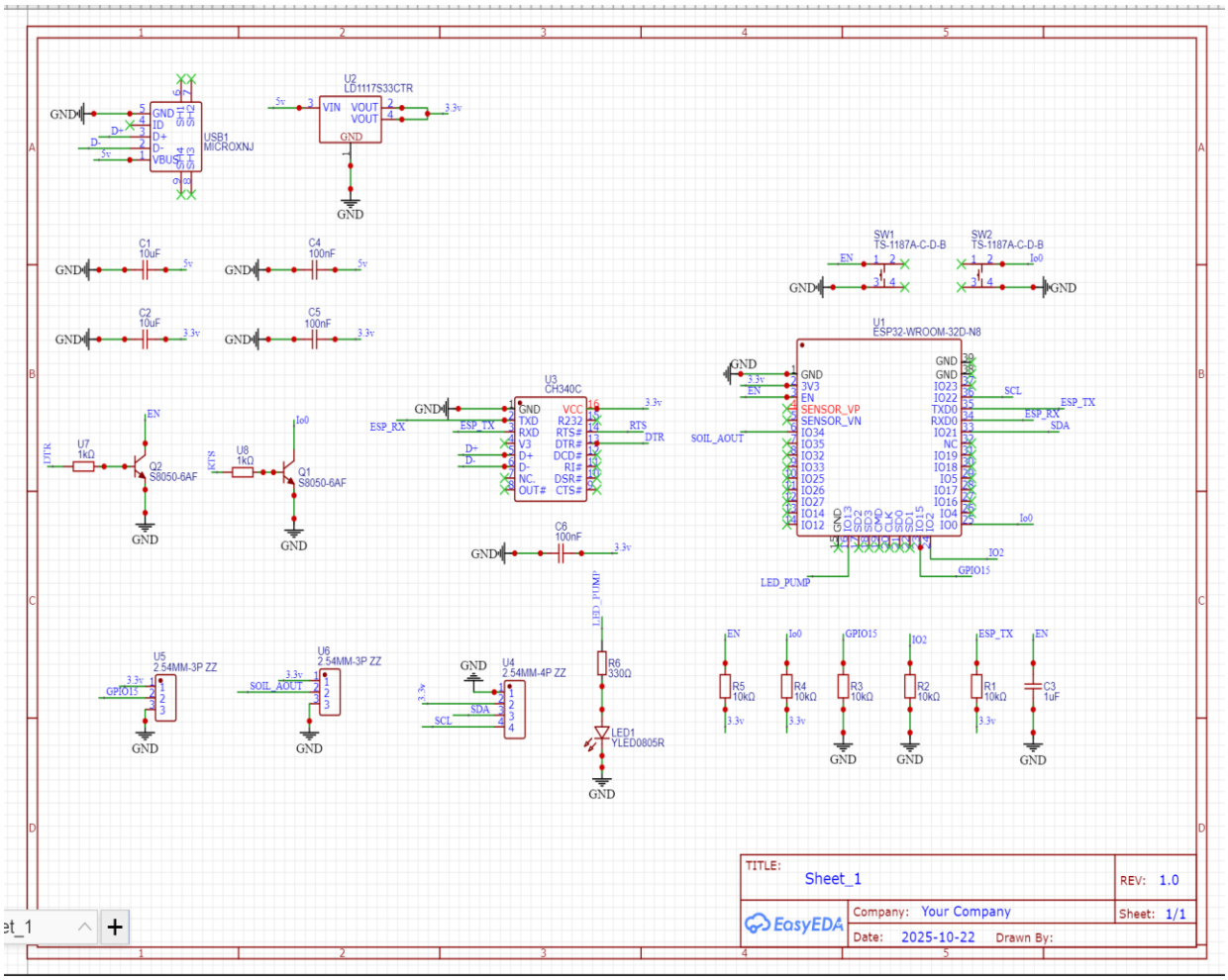


Figure_4

Phase 3: PCB Layout (The "Floor Plan")

With our schematic "blueprint" 100% complete and verified, we were ready to move from the logical *idea* to the *physical* design. This is the "floor plan" phase, where we decide the exact size, shape, and placement of every component on the actual board.

3.1. Conversion & The "Golden Rule"

1. **Conversion:** We clicked **"Design > Convert Schematic to PCB"** in EasyEDA.
2. **The Workspace:** This opened the PCB Layout Editor. It showed us a blank purple rectangle (our board, which we set to a large 100mm x 80mm workspace) and all 26 of our components in a jumbled pile.
3. **The "Rat's Nest":** The components were connected by hundreds of thin blue lines. This "rat's nest" is not a mess; it's our guide. It shows us *what* needs to be connected, based on our schematic.

Before moving a single part, we established the **Golden Rule of PCB Placement**:

"Place components that are wired together *physically close* to each other."

Why? It's Physics. Long copper "wires" (called traces) act like tiny **antennas**. They can pick up "noise" (static, radio waves) from the air, which can corrupt data and crash the ESP32. By placing related components in *tight, compact groups*, we keep the traces as short as possible. **Short traces = a stable, quiet, working board.**

3.2. The 4-Zone Placement Strategy

Our goal was to untangle the "rat's nest" by moving and **rotating (R key)** components until the blue lines were as short as possible. We did this by following our 4-Zone plan:

- **Zone 1: The Power Block (Top-Left)**
  - **What:** USB1, U2 (Regulator), C1, C2, C4, C5.
  - **Why:** This is the *most critical* group. We placed the "filters" (C1-C5) *physically touching* the "faucet" (U2 regulator). This ensures the 3.3V power is perfectly clean *before* it gets sent to the rest of the board.
- **Zone 2: The Programming Block (Middle-Left)**
  - **What:** U3 (CH340C), C6, Q1, Q2, U7, U8.
  - **Why:** The USB and programming signals are high-speed. We grouped these parts to create a short, direct path for the data (USB1 → U3 → Transistors → ESP32), which prevents data corruption.
- **Zone 3: The ESP32 Support Block (Center)**
  - **What:** U1 (ESP32) and all its buttons and resistors (R1-R5, C3, SW1, SW2).
  - **Why:** The EN (Reset) and IO0 (Boot) pins are sensitive. We placed their support parts (like R5, SW1) in the "yard" *right next* to the pins they connect to. This prevents stray noise from accidentally resetting the chip.
- **Zone 4: The I/O Block (Right & Bottom)**
  - **What:** U4, U5, U6 (headers), LED1, R6.

- **Why:** This is all about the *human*. We placed the headers on the **right edge** and the LED on the **bottom edge** so a person can easily plug in sensors and see the status light.

3.3_3D Verification & Critical Fixes

Placing components in 2D is not enough. We used the **"3D View"** to check our layout in the "real world," which instantly revealed two major flaws:

1. **The USB Port:** Our 3D view showed the USB1 port was too far inside the board. A physical cable could *not* plug in!
    - **The Fix:** We went back to the 2D view and **moved USB1** so its "mouth" (the white line on the silkscreen) was **perfectly on the edge** of the purple board outline.
2. **The R4 Resistor:** Our 2D "rat's nest" showed a long blue line for R4 stretching across the board.
    - **The Fix:** We saw that R4 (the pull-up) and SW2 (the button) both connected to the *same pin* (IO0). We **moved R4** right next to its "partner" SW2, making the blue line tiny and the layout much cleaner.

After these fixes, our component placement was 100% complete, logical, and correct.

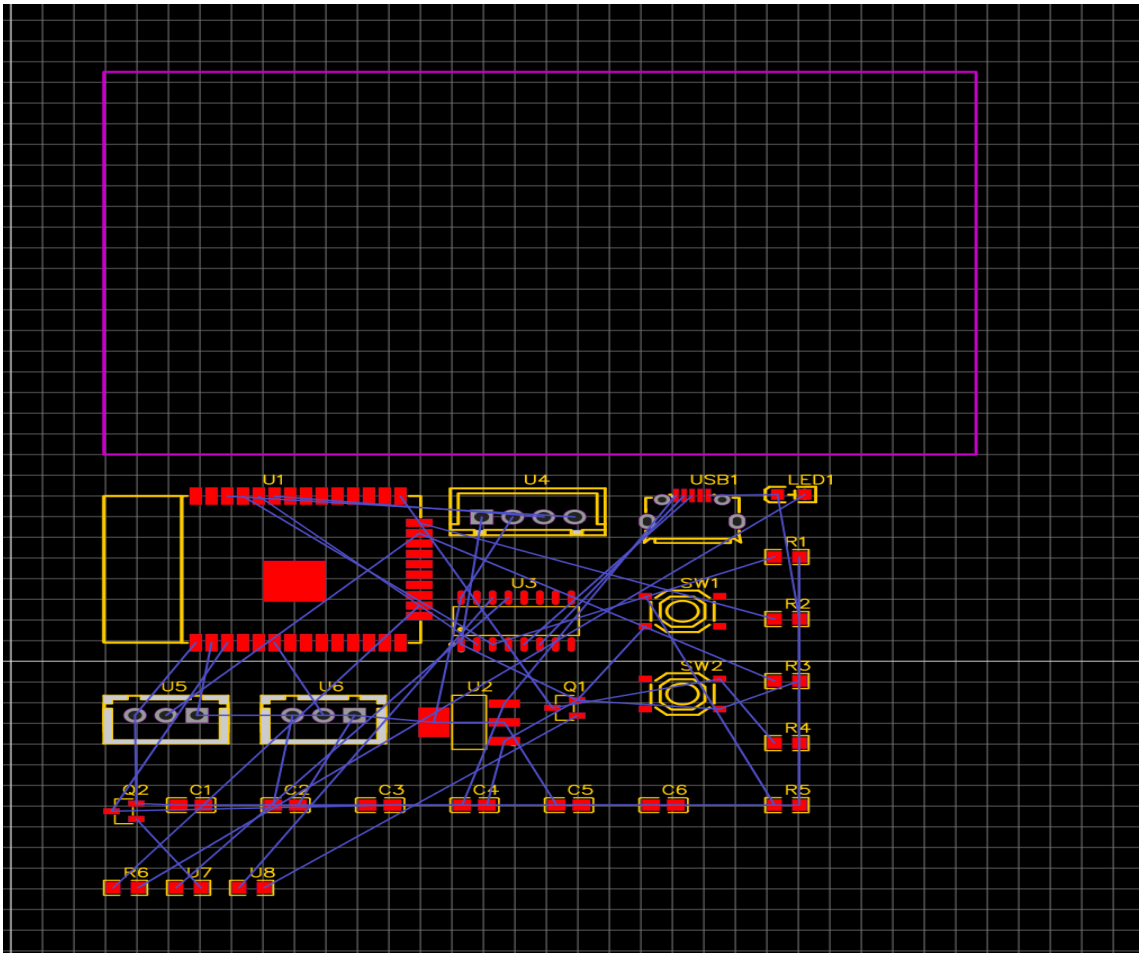


Figure_5

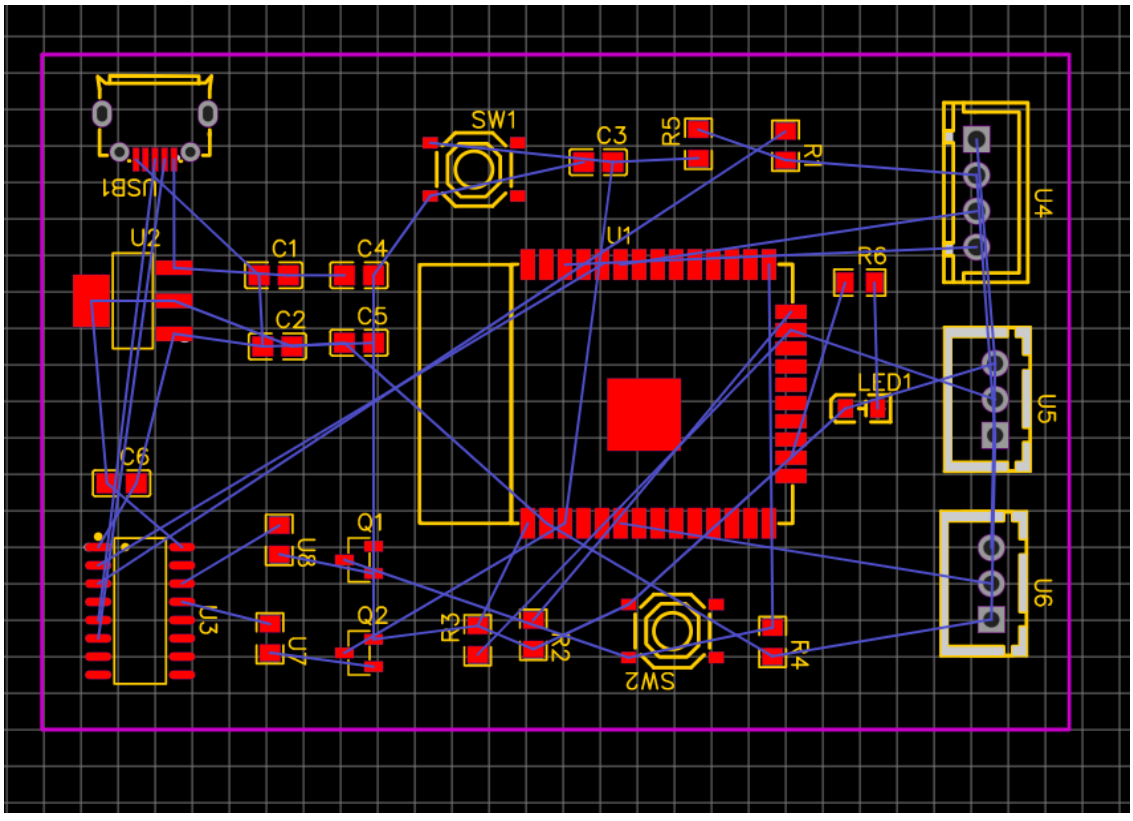


Figure_6

Phase 4: PCB Routing (Drawing the "Wires")

This was the final and most exciting phase of the design. Our "floor plan" was complete, but none of the components were actually connected. The board was still just a collection of parts and blue "rat's nest" lines.

The goal of this phase was to replace every single blue "rat's nest" line with a physical **copper "trace"** (a wire) to create the final, working circuit.

4.1. Our 2-Layer Strategy: Roads & Tunnels

Our board is a **2-layer PCB**. We had to think of it as a city with two levels of transportation:

1. **The TopLayer (Red):** The "surface roads" where all our components live. We decided to draw most of our traces here.
2. **The BottomLayer (Blue):** The "tunnels" underneath the city. We would use this layer to go *under* obstacles.
3. **Vias (The "Ramps"):** These are small, plated holes that act as "ramps" to let a trace go from the TopLayer down to the BottomLayer and back up.

4.2. Step 1: Power Routing (The "Highways")

- **Problem:** The 5v and 3.3v power lines are the most important "highways" on the board. They carry all the power. If they are too thin, they can heat up and fail.
- **Solution:**
    1. We set our **"Routing Width"** to a thick **0.5mm**. This creates a wide, robust "highway" for the electricity.
    2. We selected the **"Track"** tool (W) and the **TopLayer (Red)**.
    3. We carefully drew all the red traces to connect every 5v pad and every 3.3v pad, following the blue "rat's nest" lines.

4.3. Step 2: The Ground Plane (The "Ocean")

- **Problem:** The GND (Ground) net is the most common connection. Dozens of pins needed to connect to it. Drawing all those wires would create an impossible-to-solve mess.
- **Solution:** We used a professional technique called a **"Copper Pour"** (or "Ground Plane").
    1. We selected the **"BottomLayer" (Blue)**.
    2. We used the **"Copper Area"** tool (E) to draw a giant rectangle over our *entire* board.
    3. We assigned this giant copper area to the **GND** net.
- **Result:** The software instantly flooded the *entire* BottomLayer "basement" with a solid sheet of blue copper. This blue "ocean" automatically connects all GND pins together and acts as a massive shield, protecting our circuit from electrical noise.

4.4_Step 3: Connecting to Ground (The "Stairs")

- **Problem:** Our components' GND pads were on the "Top Floor" (Red), but our Ground "Ocean" was in the "Basement" (Blue). They weren't touching!
- **Solution:** We had to build "stairs" (Vias) to connect every GND pad down to the ocean.
    1. We selected the **"Via"** tool (P) and set its Net to GND.
    2. We placed one GND Via **right next to** *every single GND pad* on our board.
    3. We then used the **"Track"** tool (W) to draw a **short red "walkway"** from each GND pad to its personal Via.
- **Result:** All the blue "rat's nest" lines for GND vanished. Every GND pad was now successfully connected down to the blue ground plane.

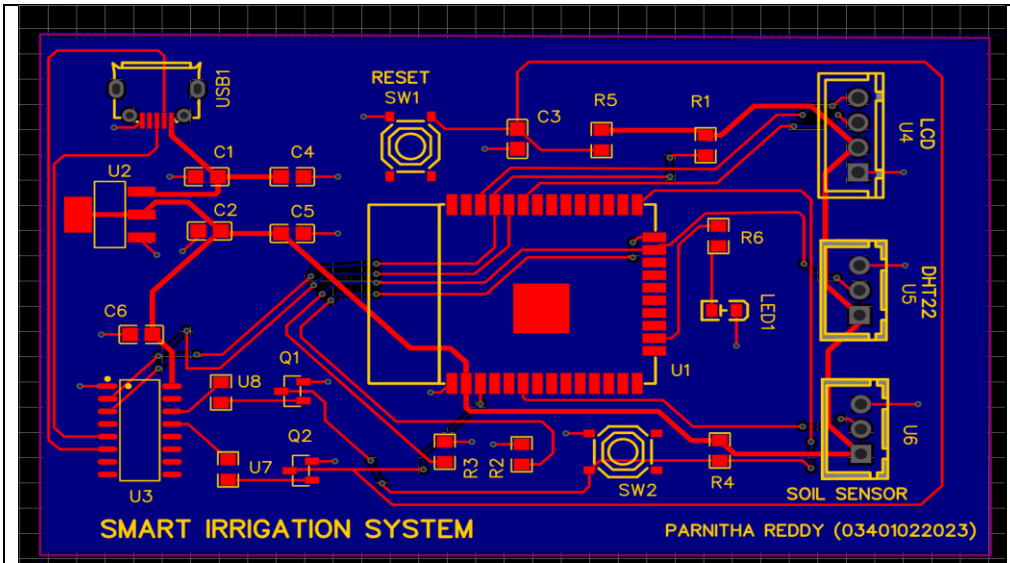
5.5. Step 4: Signal Routing (The "City Streets")

- **Problem:** With Power and Ground finished, we still had all the "signal" lines left (like ESP_TX, SDA, SCL, etc.).
- **Solution:** This was the final puzzle:
    1. We set our **"Routing Width"** back to a standard **0.254mm**.
    2. We selected the **TopLayer (Red)** and began connecting the shortest, easiest blue lines first.

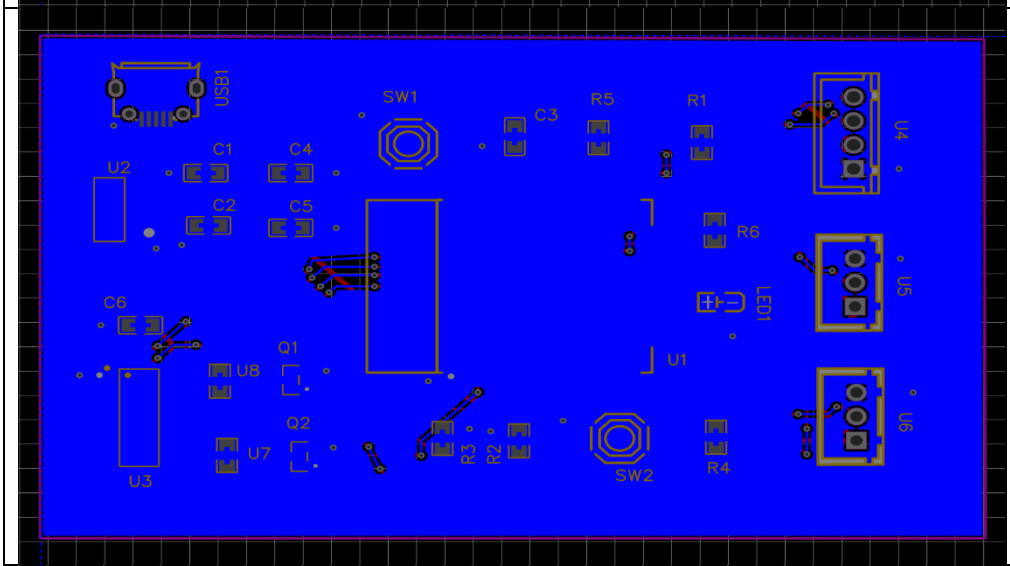
4.6_Step 5: The "Via Jump" (The "Underpass")

- **Problem:** We quickly ran into a common issue: a red trace (like ESP_RX) needed to cross a red trace we already drew (like 3.3V). You *cannot* have two red traces cross—it's a short circuit.
- **Solution:** We used the **"Via Jump"** or "Underpass" technique:
    1. We drew our **Red Trace** (the "road") right up to the obstacle.
    2. We pressed the **V key** and clicked to place a Via (a "ramp down").
    3. The tool **automatically** switched us to the **BottomLayer (Blue)**.
    4. We drew our **Blue Trace** (the "tunnel") *under* the red road.
    5. Once clear, we pressed **V** and clicked again to place another Via (a "ramp up").
    6. The tool **automatically** switched us back to the **TopLayer (Red)**, and we drew the final short red trace to our destination pad.

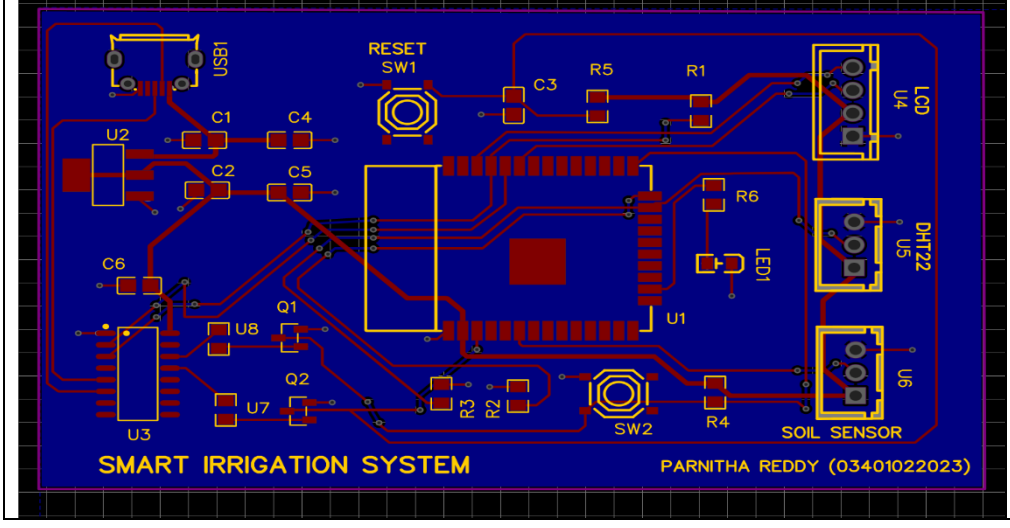
By using this "JUMP V" method repeatedly, we successfully routed 100% of the connections, solving the puzzle and eliminating *all* the remaining blue lines.



Top Layer



Bottom layer



Top Silk Layer

Phase 5: Verification & Final Output

With all the traces routed, the board *looked* finished. But a "finished" look can hide tiny, critical errors. This final phase was about **verifying** our work with automated tools and then "polishing" the design to make it look professional and ready for manufacturing.

5.1_Step 1: Design Rule Check (DRC)

- **What it is:** This is the most important check. We ran the **"Design Rule Check" (DRC)**, which is an automated "spell check" for PCBs. It scans the *entire* board to find any violations of the manufacturer's rules, such as:
    - Traces that are too close to each other (short circuit risk).
    - Traces that are too close to Vias (short circuit risk).
    - Traces that are too close to the board edge.
- **The Result (46 Errors!):** Our first DRC report was terrifying—it listed **46 errors!**
- **The Fix (One Simple Command):** We realized that all 46 errors were the same problem. The **blue GND copper pour** (the "ocean") was "stale." It hadn't updated since we added all our red and blue traces.

    1. We pressed **Shift + B** on the keyboard. This is the **"Rebuild Copper Area"** command.
    2. This forced the blue GND pour to "re-flow" around all our new traces, automatically creating a safe clearance gap.
    3. This single command **fixed 44 of the 46 errors instantly!**

- **Fixing the Last Two Errors:** We were left with two *real* errors, which the DRC highlighted with **yellow "X"s**. We clicked on each error, which zoomed us in. We saw a blue trace was slightly too close to a Via.
    - **The Fix:** We simply selected the **"Track"** tool (W) and **"nudged"** (clicked and dragged) the blue trace a tiny bit until the yellow "X" disappeared.
- **Final Result:** After nudging the two traces and pressing Shift + B one last time, we re-ran the DRC. The result was **"DRC Errors (0)"**. Our board was now 100% verified and error-free.

5.2_Step 2: Final Polish (Silkscreen & Outline)

The board was electrically perfect, but it didn't look professional yet.

1. **Cleaning the Silkscreen (Yellow Text):** The component names (like U1, C3, R5) were messy and overlapping our red solder pads.
    - **Action:** We selected the **"TopSilkLayer"** (Yellow). We used the **"Text"** tool (T) to add helpful labels like LCD, DHT22, SOIL SENSOR, and RESET.
    - We also moved all the default labels (U1, C3, etc.) and our new text into clean, empty blue spaces, ensuring no yellow text was on top of a red pad.
    - Finally, we added our project title and name to the board.

2. **Shrinking the Board Outline (Saving Money):** Our components were in a small, compact area, but we were still using the large 100mm x 80mm board outline.
    - **Action:** We selected the **"BoardOutline"** layer (purple) and simply **dragged the purple edges inward**. We left a 3-5mm margin around our components,creating a snug, custom-fit board. This makes the final product cheaper to manufacture.
    - **Final Rebuild:** After shrinking the outline, we pressed **Shift + B** one last time. This made the blue GND pour "re-flow" and perfectly fill the new, smaller board shape.

5.3_ Step 3: Gerber File Generation (The Final Product)

The board was 100% complete, correct, and polished. It was time to generate the "master files" for the factory.

1. We clicked **Fabrication > PCB Fabrication File (Gerber)**.
2. In the pop-up window, we confirmed the settings and clicked **"Generate Gerber"**.
3. After a moment, a preview of our board appeared. We reviewed it one last time.
4. We clicked **"Download Gerber Files"**.

This saved a single **.zip file** to our computer. This file is the final product of our entire design process. It contains all the individual layer instructions (copper, solder mask, silkscreen, drill holes) that any PCB manufacturer in the world (like JLCPCB) needs to build our physical, custom-designed board.

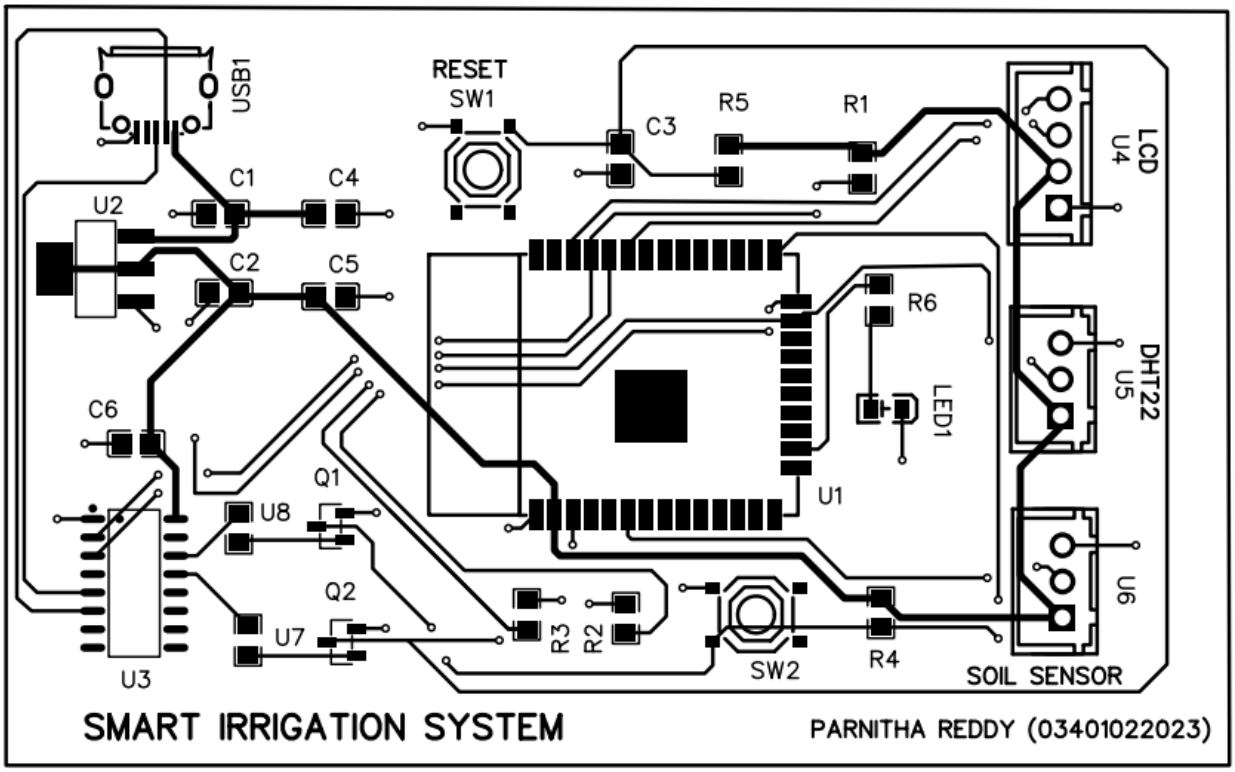
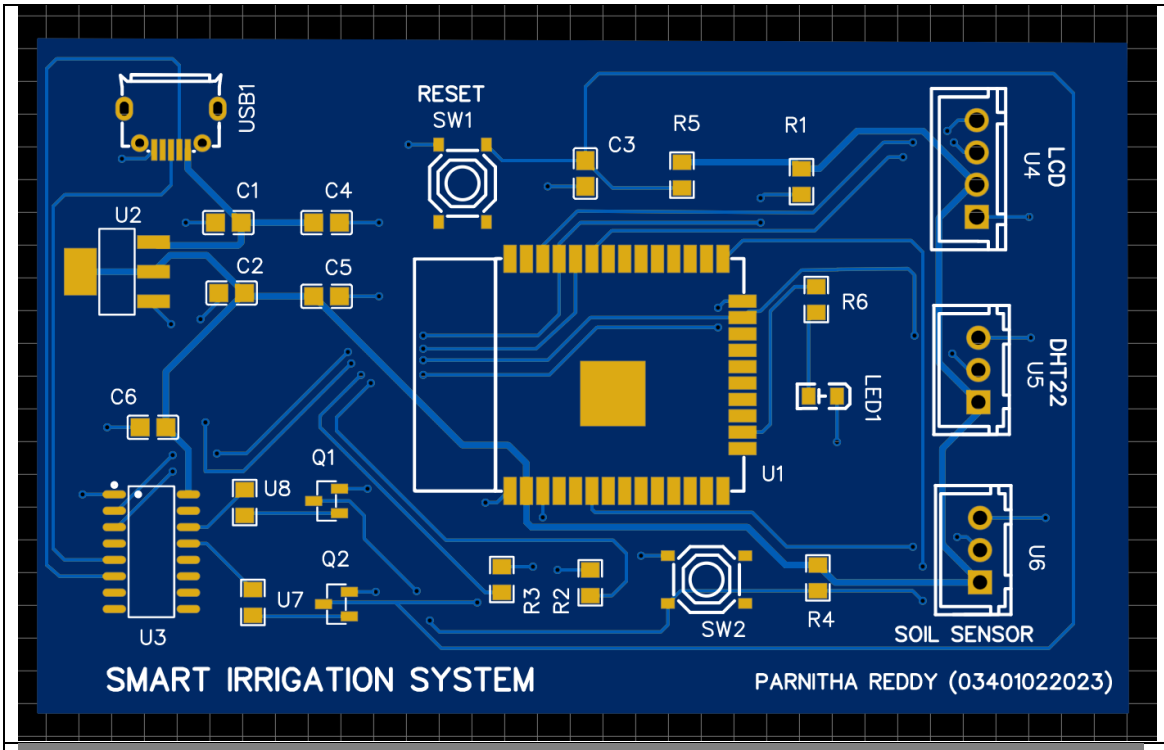
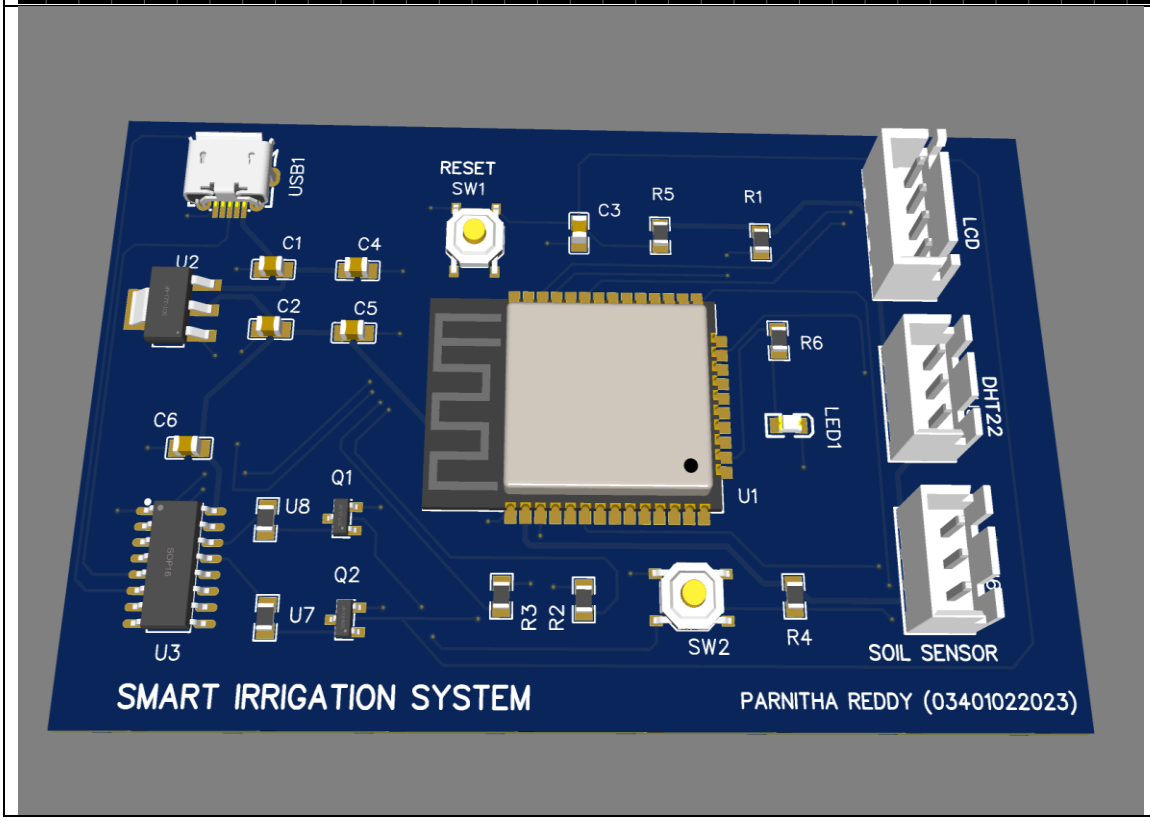

USB1
RESET
SW1
C3
R5
R1
LCD
U4
C1
C4
U2
C2
C5
R6
DHT22
U5
LED1
C6
Q1
U8
U1
U6
Q2
U7
R3
R2
SW2
R4
SOIL SENSOR
U3
SMART IRRIGATION SYSTEM
PARNITHA REDDY (03401022023)

Figure_7

 USB1 RESET SW1 C3 R5 R1 LCD U4 C1 C4 U2 C2 C5 R6 DHT22 U5 LED1 C6 Q1 U8 U1 Q2 U7 R3 R2 SW2 R4 U6 U3 SOIL SENSOR SMART IRRIGATION SYSTEM PARNITHA REDDY (03401022023)	**2D**
 USB1 RESET SW1 C3 R5 R1 LCD C1 C4 U2 C2 C5 R6 DHT22 LED1 C6 Q1 U8 U1 Q2 U7 R3 R2 SW2 R4 U3 SOIL SENSOR SMART IRRIGATION SYSTEM PARNITHA REDDY (03401022023)	**3D**

Conclusion & Next Steps

Project Summary

This project successfully achieved its goal of designing a custom, from-scratch Printed Circuit Board for an ESP32-based "Smart Irrigation System." The entire design process was documented, from the initial **software simulation** in Wokwi (Phase 1) to the creation of a **formal schematic** (Phase 2). This was followed by a complete **physical PCB layout** (Phase 3), full **2-layer routing** (Phase 4), and a final **verification and polish** (Phase 5).

The final deliverable is a single, error-free **Gerber .zip file**. This file contains all the necessary manufacturing data to produce a professional, compact, and fully functional 2-layer PCB.

Key Skills Demonstrated

This project was a comprehensive exercise in systems integration, demonstrating proficiency in:

- **System Design:** Translating a software prototype into a robust hardware solution.
- **Schematic Capture:** Designing complex support circuits for power, programming, and booting a microcontroller.
- **PCB Layout:** Applying the "Golden Rule" of component placement to create a compact, low-noise, and manufacturable 4-zone layout.
- **2-Layer Routing:** Using industry-standard techniques like ground planes, Vias, and "underpass" routing to solve complex layout puzzles.
- **Verification (DRC):** Using automated tools to find and fix 100% of all design rule errors, ensuring a viable final product.

Next Steps

The design phase is 100% complete. The logical next steps for this project are:

1. **Fabrication:** Send the generated Gerber file to a PCB manufacturer (like JLCPCB) to have the bare boards produced.
2. **Assembly:** Procure all 26 components from the Bill of Materials and hand-solder them onto the manufactured PCB.
3. **Testing & Deployment:** Upload the original Wokwi code to the physical board and test its functionality in a real-world environment with a physical plant, soil, and water pump.

Bibliography & References

Component Datasheets

- **Espressif Systems**, "ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet," Version 1.8, 2021. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf
- **Nanjing Qinheng Microelectronics (WCH)**, "CH340 Datasheet (USB to Serial Chip)," Version 1, 2011. [Online]. Available: https://www.mpja.com/download/35227cpdata.pdf
- **STMicroelectronics**, "LD1117S33CTR - Low Drop Fixed and Adjustable Positive Voltage Regulators," August 2021. [Online]. Available: https://www.st.com/resource/en/datasheet/ld1117.pdf
- **Aosong Electronics**, "DHT22 (AM2302) Humidity and Temperature Sensor," User Manual, Version 1.2. [Online]. Available: https://akizukidenshi.com/download/ds/aosong/AM2302.pdf

Software & Design Tools

- **EasyEDA**, "EasyEDA Tutorial & User Manual," [Online]. Available: https://docs.easyeda.com/
- **Wokwi**, "Wokwi ESP32 Simulator Documentation," [Online]. Available: https://docs.wokwi.com/

Technical Standards & References

- **IPC Association**, "IPC-2221A: Generic Standard on Printed Board Design," May 2003. *(Note: This is the standard industrial rulebook for things like trace width and clearance that you used in your design).*
- **Espressif Systems**, "ESP32 Hardware Design Guidelines," Version 3.3, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_hardware_design_guidelines_en.pdf