BABU BANARSI DAS University

Internet of Things Project File

(NITBC4351)

SUBMITTED BY:
Adarsh Kumar | Abhinav Singh
|Netraja | Sakshi Chaubey|
Class: IOTBC2

SUBMITTED TO:
Mr. Vikas Kumar

Project

Aim- To design a Smart Energy Monitoring and Theft Detection system using IoT that measures supply and load current, detects abnormal usage or theft, and alerts the user through display and buzzer.

Outcomes/Learning:

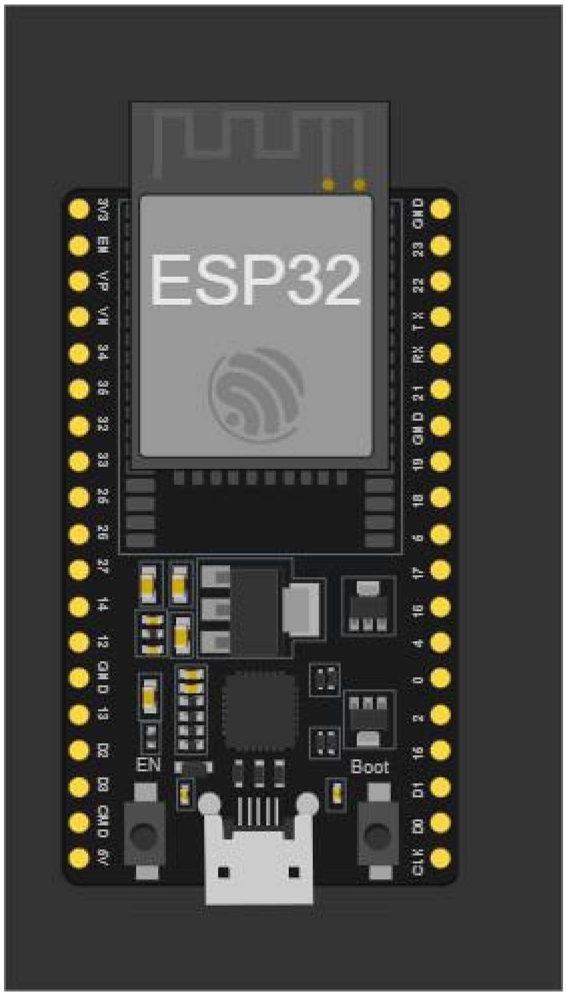
1) Detects theft and bypass conditions accurately.
2) Displays real-time supply and load readings on LCD.
3) Sends data to IoT dashboard for remote monitoring.
4) Activates buzzer and LED during theft conditions.

Required Tool:

1) ESP32 Microcontroller
2) 2 × CT Sensors (30A)
3) 16×2 LCD Display (I2C)
4) LED
5) Buzzer
6) Push Button
7) Breadboard
8) Jumper Wires (M–M, F–F, M–F)
9) Resistor
10) Extension Board

Working: Two CT sensors measure the supply and load current.
The ESP32 compares both readings — if the difference exceeds 0.6A or the supply is zero while load is active, theft or bypass is detected.
In normal conditions, the LCD shows live readings and sends data to the IoT platform.
During theft, the buzzer and LED turn ON to indicate alert.

Tool 1: ESP32 Micro-controller : The ESP32 is a micro controller with built-in Wi-Fi and Bluetooth used for IoT and embedded projects.



1. 3V3 (Pin 1): Provides 3.3V power output for components.

2. GND (Multiple Pins – e.g., 2, 14, 25, etc.): Connects to the circuit's ground.

3. EN (Pin 3): Enables or resets the ESP32 when toggled.

4. GPIO Pins (e.g., GPIO0–GPIO39): Used for digital input and output operations.

5. ADC Pins (GPIO32–GPIO39): Read analog sensor values.

6. DAC Pins (GPIO25, GPIO26): Generate analog output signals.

7. TX/RX Pins (TX0=GPIO1, RX0=GPIO3): Handle serial communication

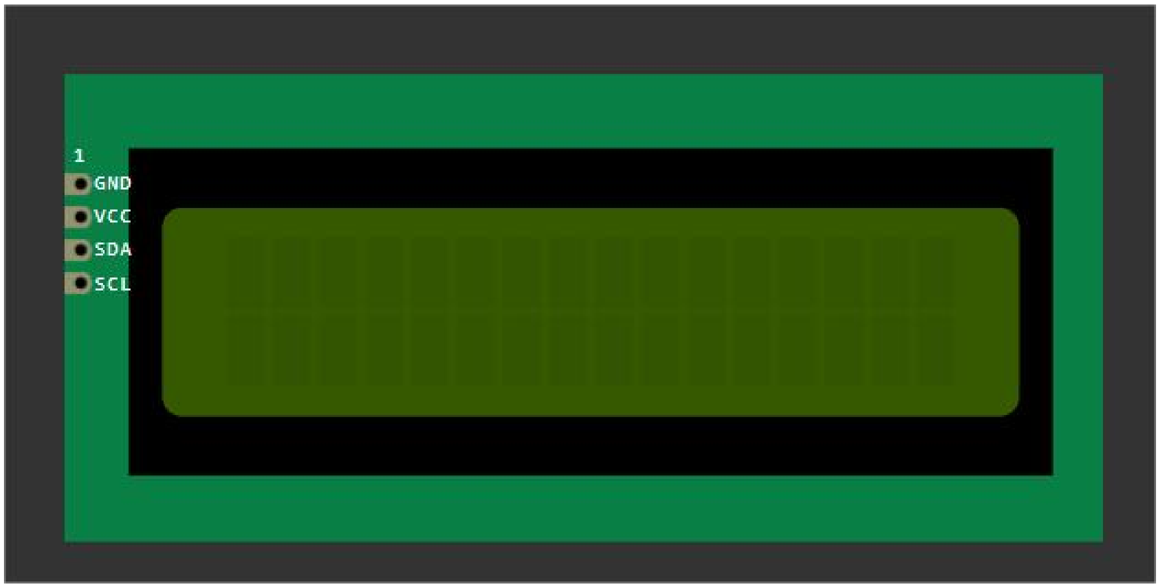
(transmit/receive).

8. SDA/SCL Pins (SDA=GPIO21, SCL=GPIO22): Used for I²C communication.

9. MOSI/MISO/SCK/CS Pins (MOSI=GPIO23, MISO=GPIO19, SCK=GPIO18, CS=GPIO5): Used for SPI communication.

Tool 2- CT Sensors (30A) : A CT sensor 30A measures alternating current by detecting the magnetic field generated around a conductor.

Tool 3 : 16×2 LCD Display (I2C) : A 16×2 LCD with I2C displays two lines of text (16 characters each) and communicates using only two data wires for easier connections.



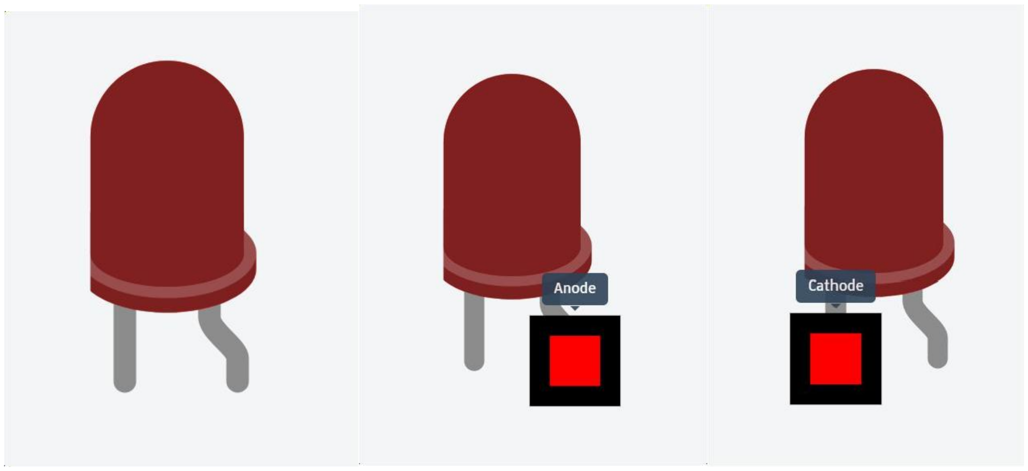
VCC: Supplies power to the LCD module.

GND: Connects to the circuit's ground.

SDA: Transfers data between the LCD and micro-controller.

SCL: Carries the clock signal for I2C communication.

Tool 4: LED :A light-emitting diode that glows when current passes through it.

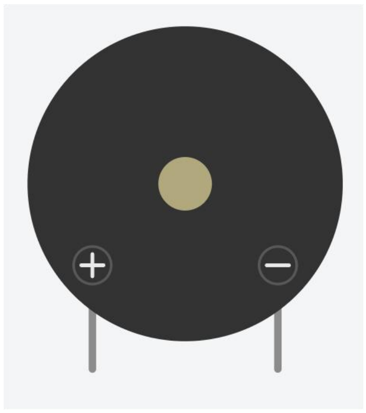
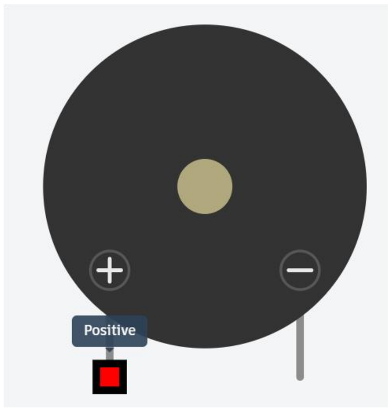


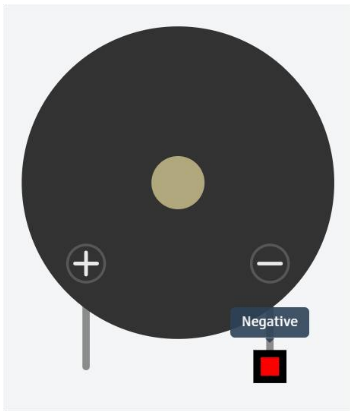
Anode (+): The longer leg of the LED. It is the positive terminal where

current enters.

Cathode (–): The shorter leg of the LED. It is the negative terminal where current exits.

Tool 5: Buzzer : A buzzer converts electrical signals into sound to provide audible alerts or indications.

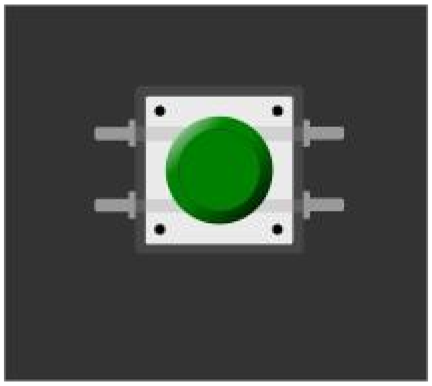





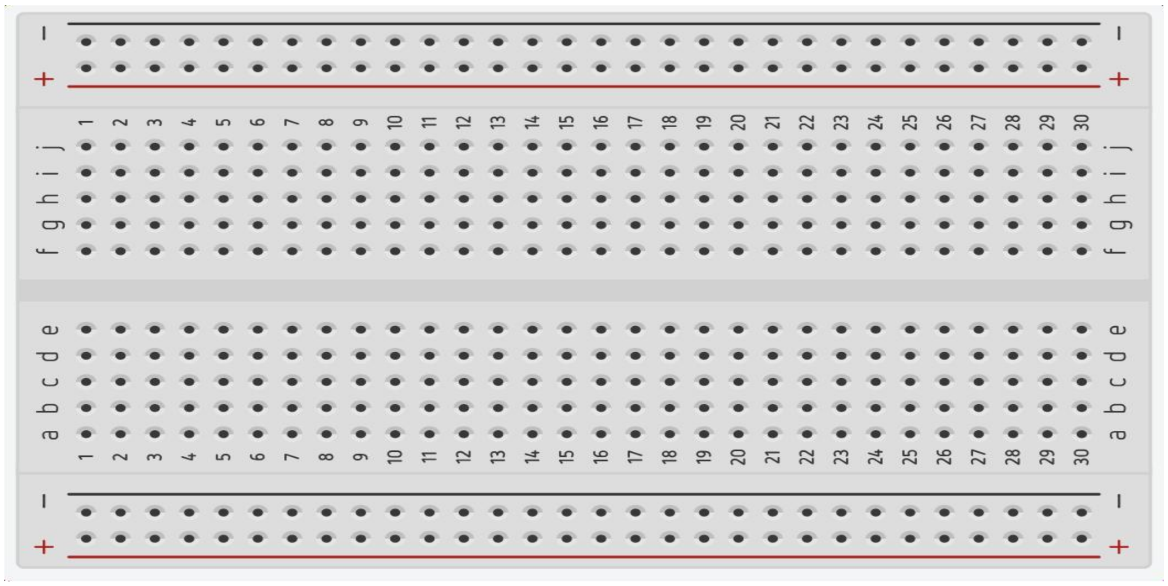
Positive pin: Connects to the power supply to activate the buzzer.

Negative pin: Connects to ground to complete the circuit.

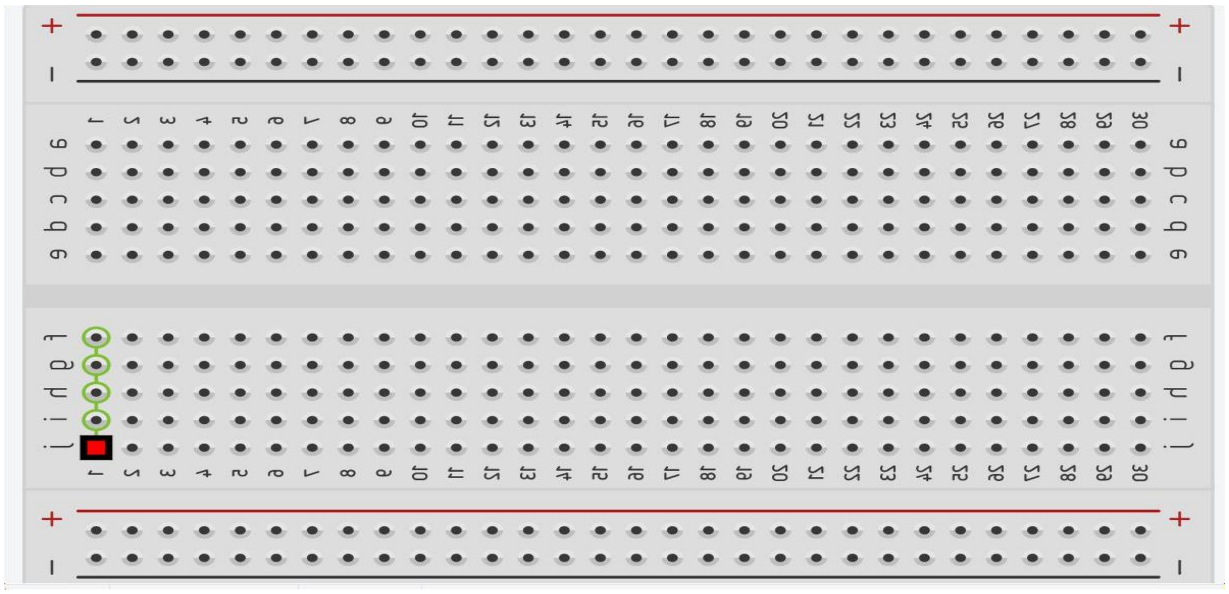
Tool 6 : Push Button : A push button is a simple switch that connects or disconnects a circuit when pressed.



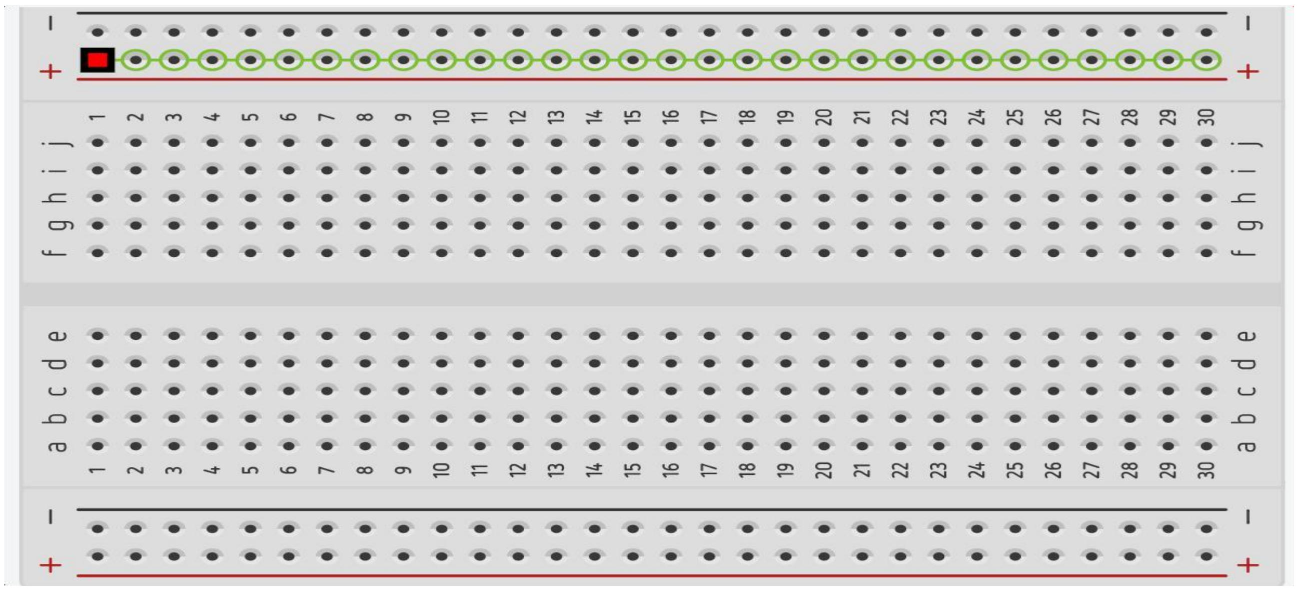
Tool 7: Breadboard: A board used to connect electronic components without soldering.



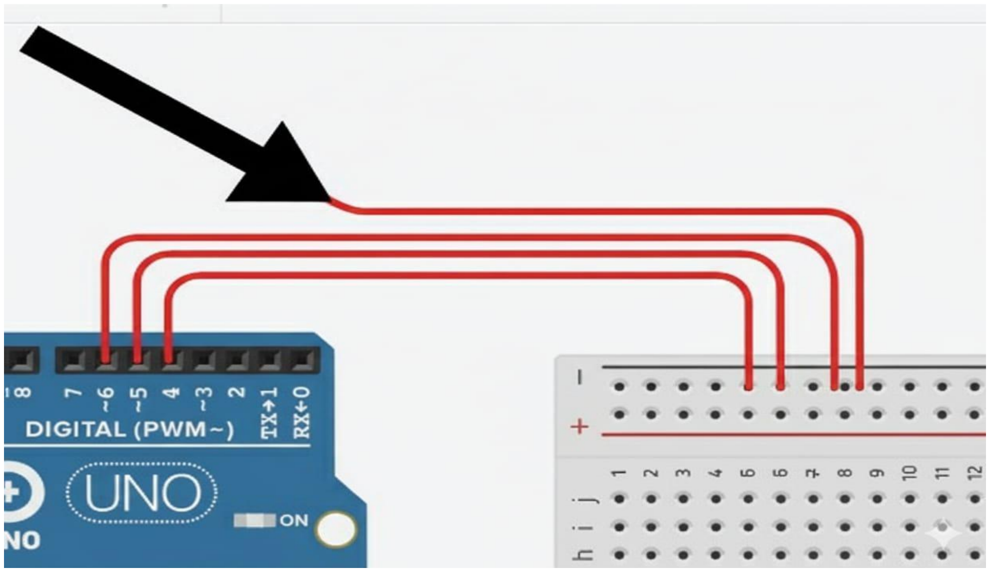
Vertical Lines: Side rails for power supply (+ and –).



Horizontal Lines: Middle rows where components are connected.

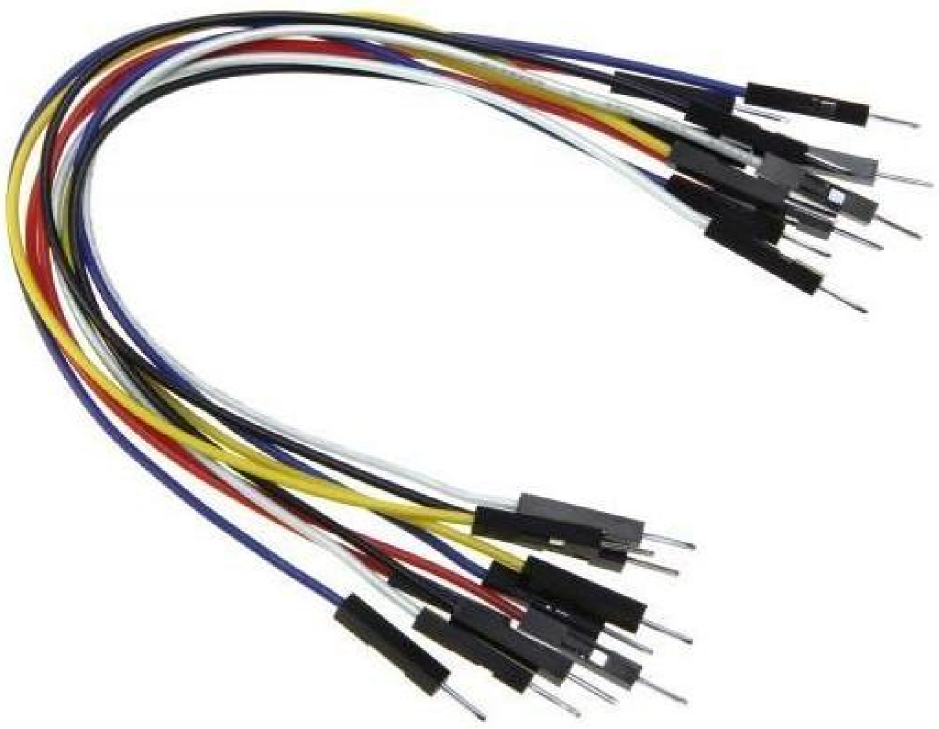


Tool 8: Jumper Wires : **Jumper Wires: Small connecting wires used to make connections between components and the breadboard/Arduino.**

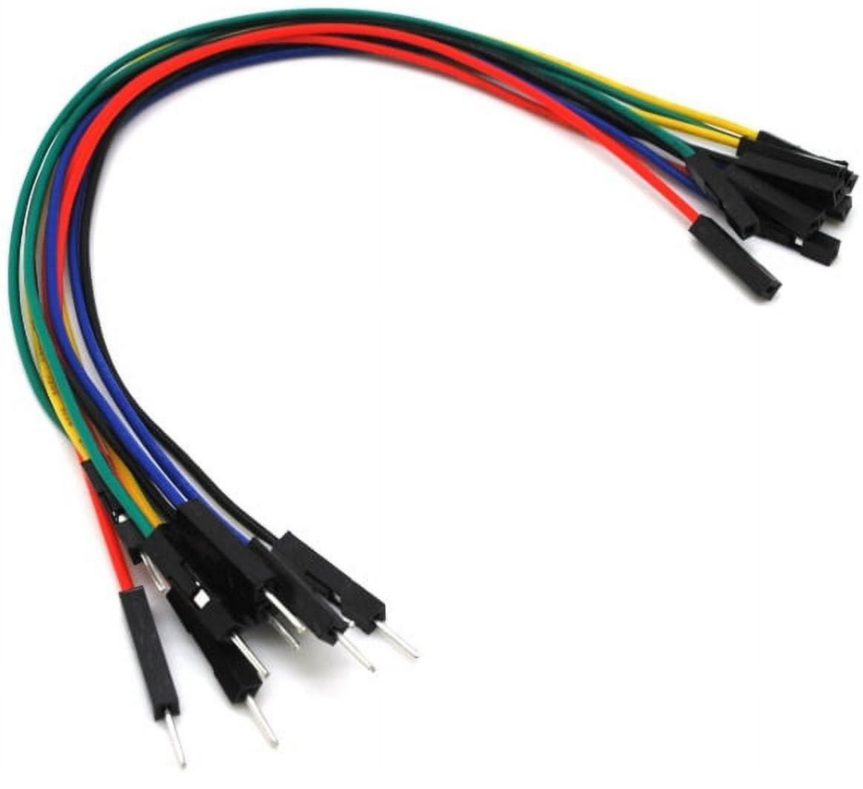


Types of Jumping Wires:

1-Male to Male Jumper Wires: Used to connect two points on a breadboard from Arduino pins to the breadboard.

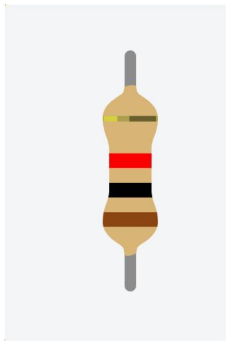


2-Male to Female Jumper Wires: Used to connect Arduino pins (male) to components with female headers

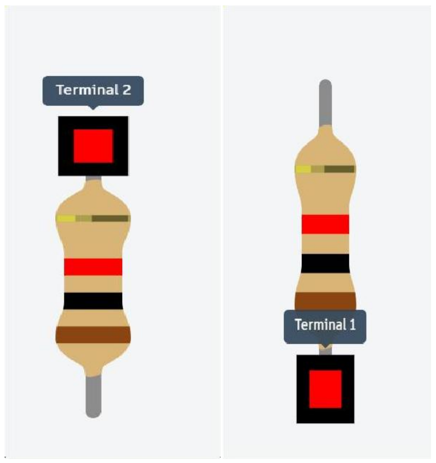


3-Female to Female Jumper Wires: Used to connect two components or modules that both have male pins.

Tool 9: Resistor: A resistor is an electronic component that opposes or limits the flow of electric current.



Terminal 1 & 2: Terminals are the two connection points of a component (like a resistor) where current enters and leaves.

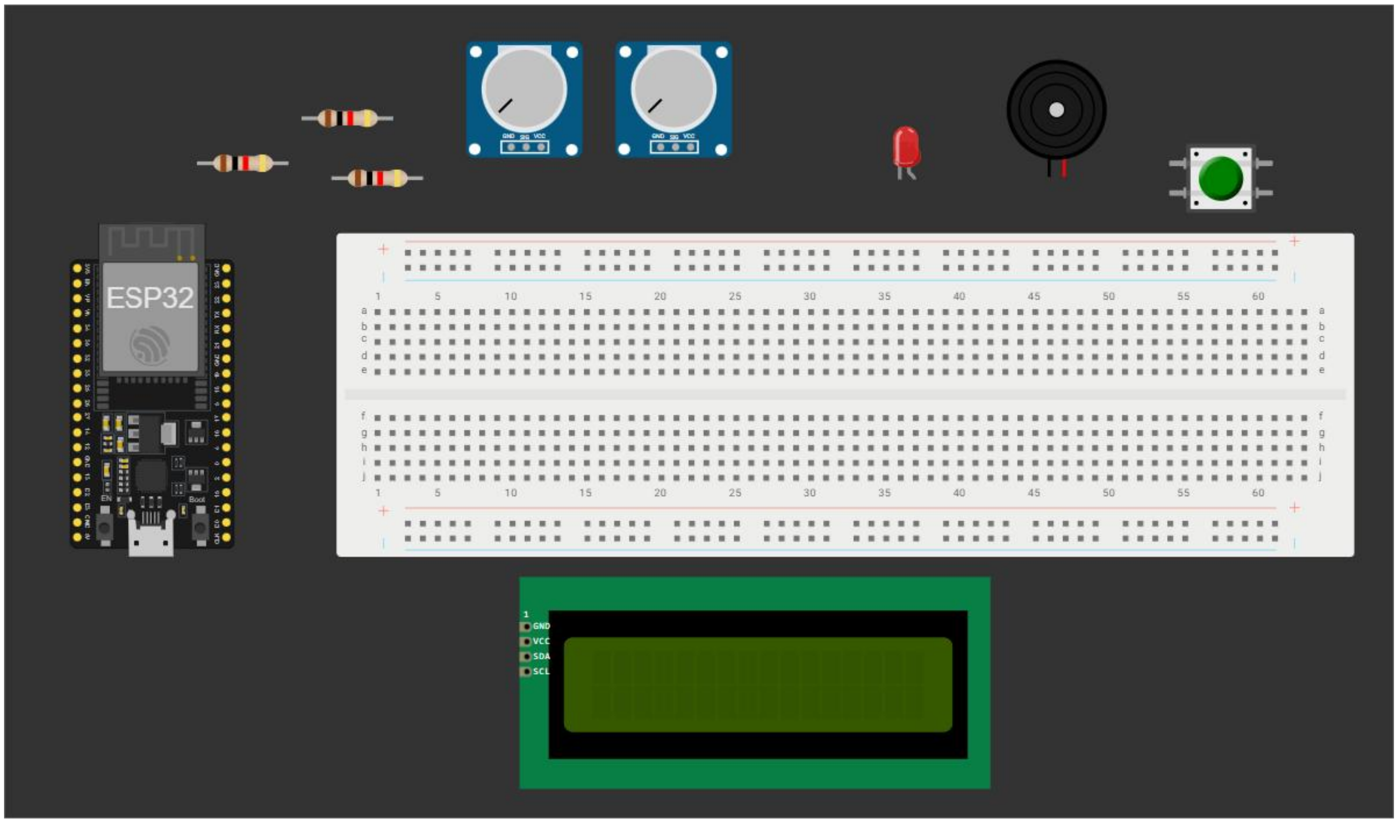


Tool 10 : Extension Boards : An extension board provides multiple power outlets from a single socket to connect several devices at once.

Working (in Wokwi Simulator) :

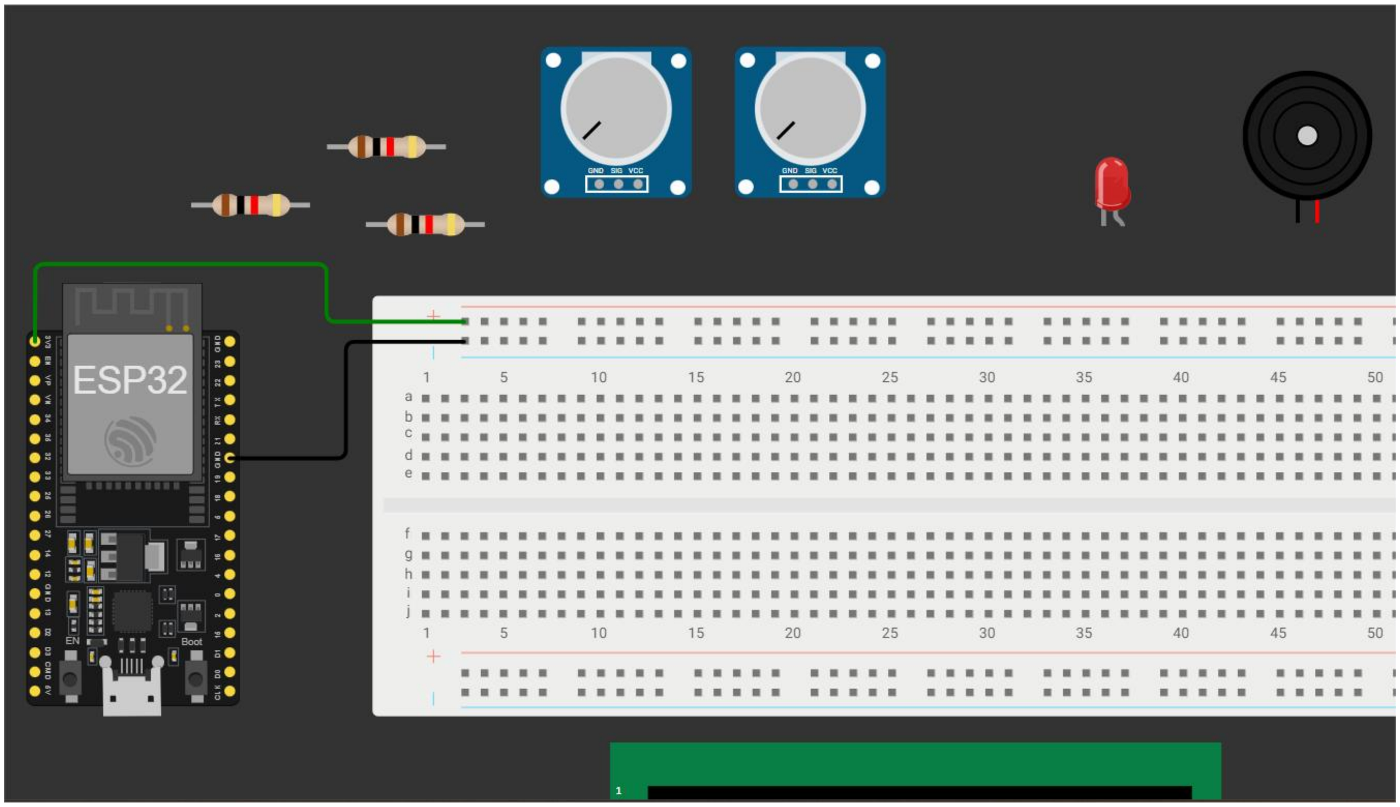
Note : Since Wokwi doesn't provide CT Sensors so, we will use Potentiometer to Demonstrate the project.

Step 1 : Gather all the required components (ESP32, Breadboard, Potentiometers, LCD 16×2(I2C), Buzzer, LED, Resistor, Push Button)



Step 2- Make the necessary Connections with jumper wire.

Make common 3v3 and GND >> connect 3v3 to +ve rail of horizontal line of breadboard and GND to -ve rail of breadboard .



Also make 5v to positive rail of horizontal line below in Breadboard. Do same thing with GND.

Connections --

1. Potentiometer (Supply Sensor)

SIG → GPIO 34
VCC → 3.3 V rail
GND → GND rail

2. Potentiometer (Load Sensor)

SIG → GPIO 35
VCC → 3.3 V rail
GND → GND rail

3. LED

Anode (+) → 220 Ω resistor → GPIO 25

Cathode (–) → GND rail

4. Buzzer

Positive (+) → GPIO 26
Negative (–) → GND rail

5. Push Button

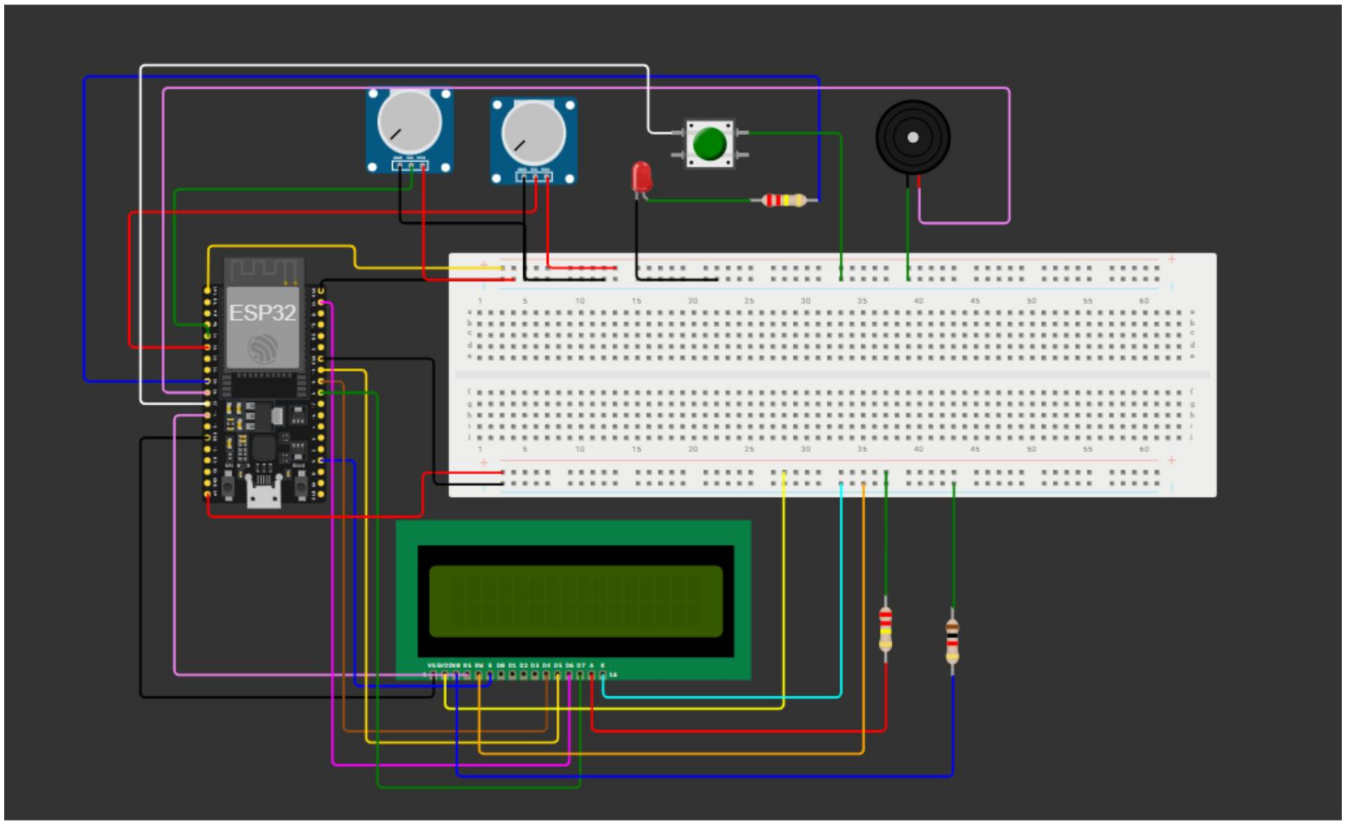
One leg → GPIO 27
Other leg → GND rail

6. 16×2 LCD (Without I²C)

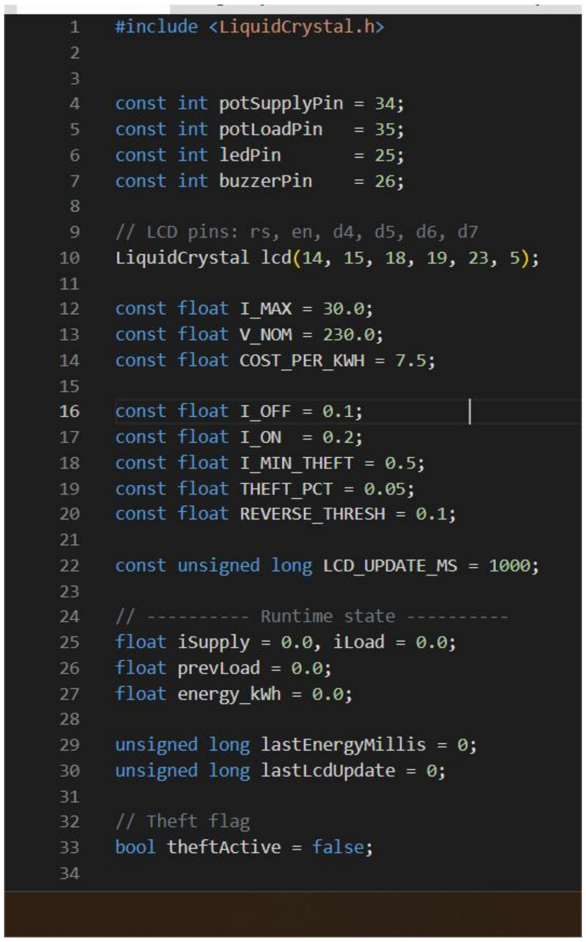
VSS → GND rail
VDD → 5 V rail
V0 → GND via resistor (contrast control)
RS → GPIO 14
RW → GND rail
E → GPIO 15
D4 → GPIO 18
D5 → GPIO 19
D6 → GPIO 23
D7 → GPIO 5
A (Backlight +) → 5 V via 220 Ω resistor
K (Backlight –) → GND rail

7. Power Rails

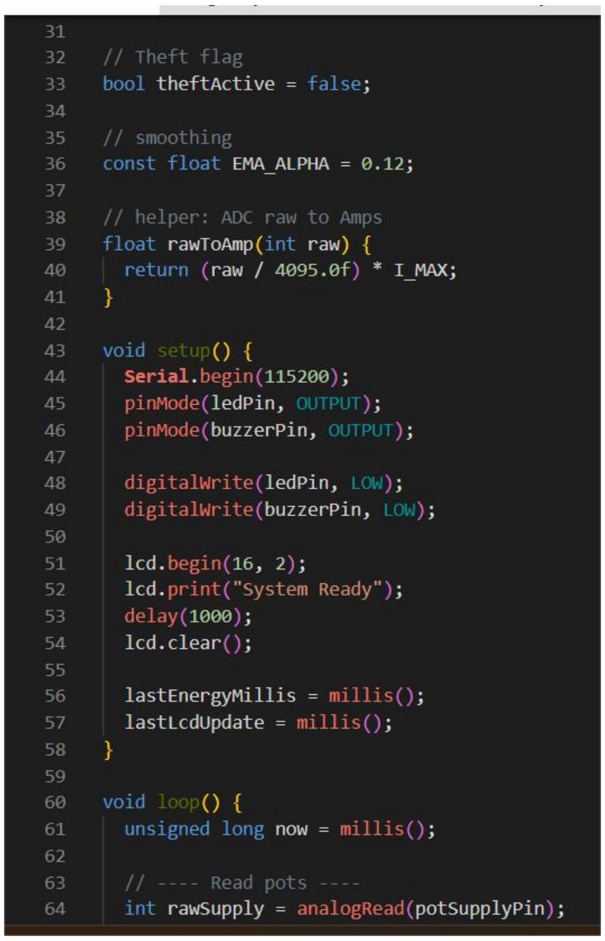
3.3 V rail → ESP32 3V3 pin (for sensors)
5 V rail → ESP32 VIN pin (for LCD)
GND rail → ESP32 GND (common ground)



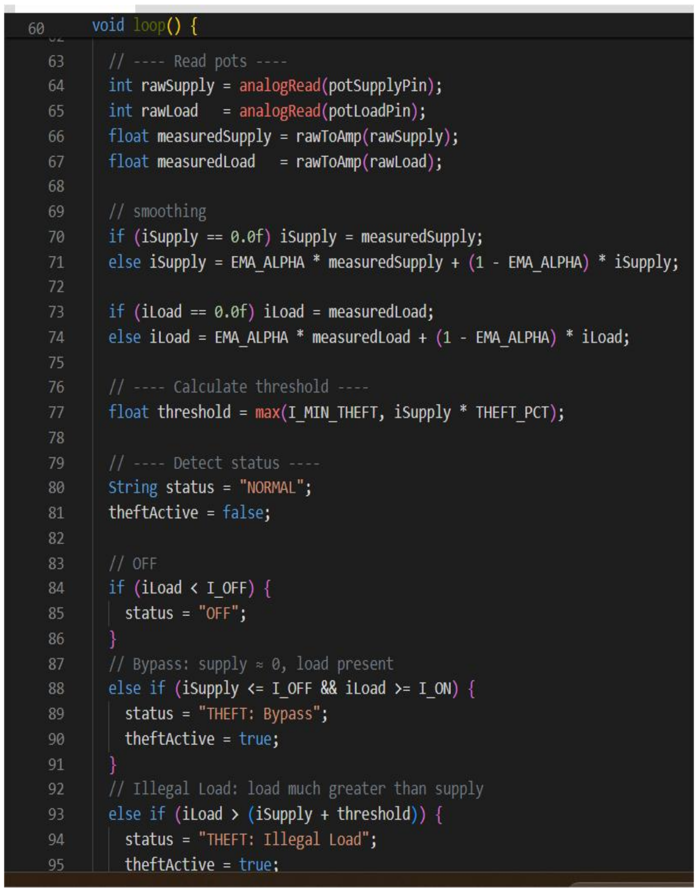
Step 3 - Upload the Code -



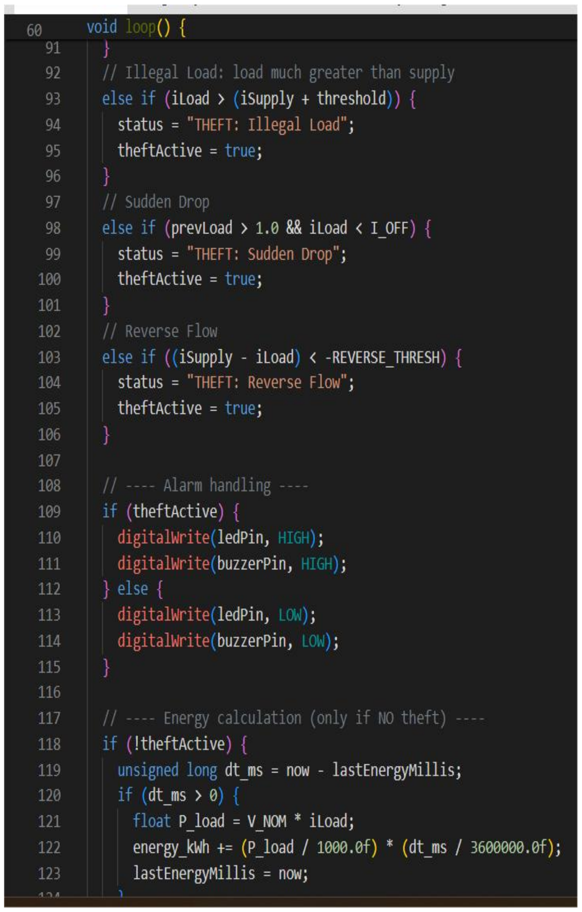
```cpp
#include <LiquidCrystal.h>


const int potSupplyPin = 34;
const int potLoadPin   = 35;
const int ledPin       = 25;
const int buzzerPin    = 26;

// LCD pins: rs, en, d4, d5, d6, d7
LiquidCrystal lcd(14, 15, 18, 19, 23, 5);

const float I_MAX = 30.0;
const float V_NOM = 230.0;
const float COST_PER_KWH = 7.5;

const float I_OFF = 0.1;
const float I_ON  = 0.2;
const float I_MIN_THEFT = 0.5;
const float THEFT_PCT = 0.05;
const float REVERSE_THRESH = 0.1;

const unsigned long LCD_UPDATE_MS = 1000;

// ---------- Runtime state ----------
float iSupply = 0.0, iLoad = 0.0;
float prevLoad = 0.0;
float energy_kWh = 0.0;

unsigned long lastEnergyMillis = 0;
unsigned long lastLcdUpdate = 0;

// Theft flag
bool theftActive = false;

```



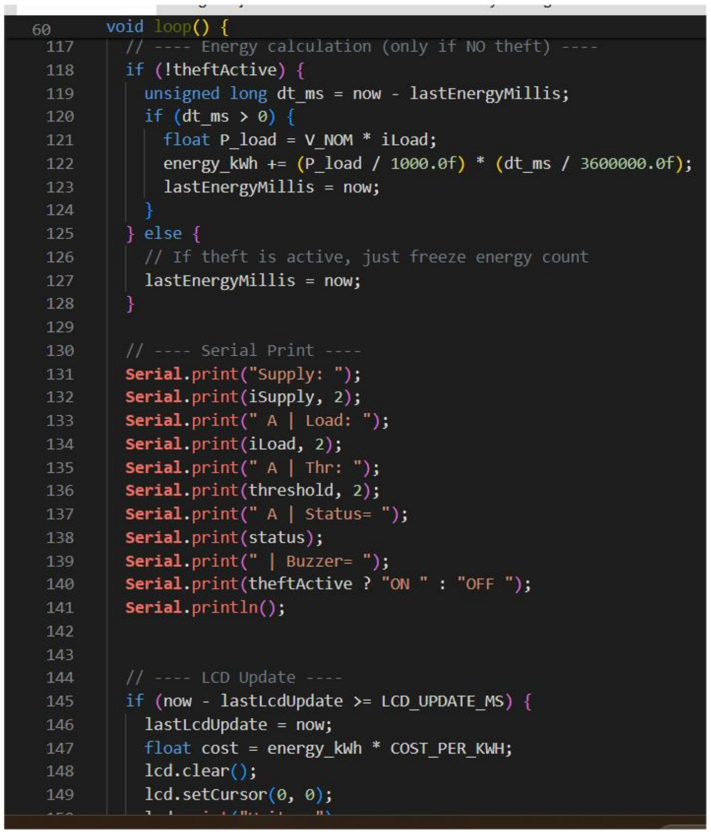
```cpp

// Theft flag
bool theftActive = false;

// smoothing
const float EMA_ALPHA = 0.12;

// helper: ADC raw to Amps
float rawToAmp(int raw) {
  return (raw / 4095.0f) * I_MAX;
}

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  digitalWrite(ledPin, LOW);
  digitalWrite(buzzerPin, LOW);

  lcd.begin(16, 2);
  lcd.print("System Ready");
  delay(1000);
  lcd.clear();

  lastEnergyMillis = millis();
  lastLcdUpdate = millis();
}

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

  // ---- Read pots ----
  int rawSupply = analogRead(potSupplyPin);
```



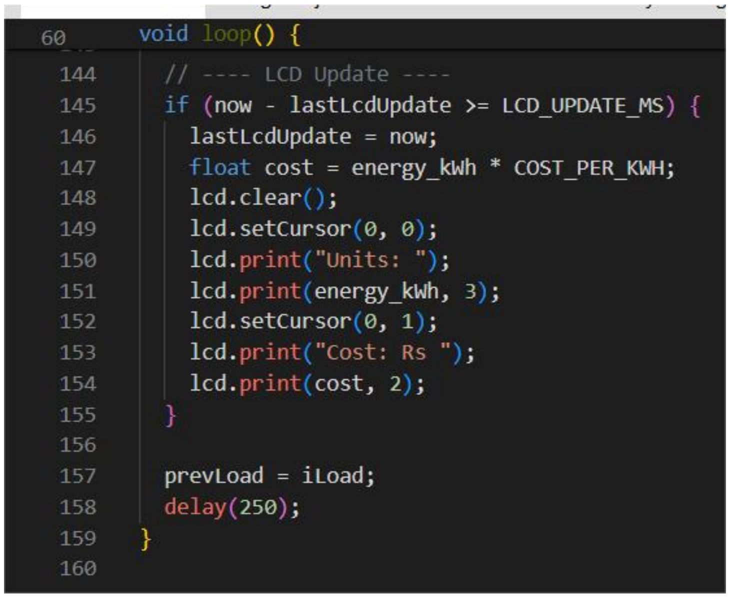
```cpp
void loop() {
  // ---- Read pots ----
  int rawSupply = analogRead(potSupplyPin);
  int rawLoad   = analogRead(potLoadPin);
  float measuredSupply = rawToAmp(rawSupply);
  float measuredLoad   = rawToAmp(rawLoad);

  // smoothing
  if (iSupply == 0.0f) iSupply = measuredSupply;
  else iSupply = EMA_ALPHA * measuredSupply + (1 - EMA_ALPHA) * iSupply;

  if (iLoad == 0.0f) iLoad = measuredLoad;
  else iLoad = EMA_ALPHA * measuredLoad + (1 - EMA_ALPHA) * iLoad;

  // ---- Calculate threshold ----
  float threshold = max(I_MIN_THEFT, iSupply * THEFT_PCT);

  // ---- Detect status ----
  String status = "NORMAL";
  theftActive = false;

  // OFF
  if (iLoad < I_OFF) {
    status = "OFF";
  }
  // Bypass: supply ≈ 0, load present
  else if (iSupply <= I_OFF && iLoad >= I_ON) {
    status = "THEFT: Bypass";
    theftActive = true;
  }
  // Illegal Load: load much greater than supply
  else if (iLoad > (iSupply + threshold)) {
    status = "THEFT: Illegal Load";
    theftActive = true;
```



```cpp
void loop() {
  }
  // Illegal Load: load much greater than supply
  else if (iLoad > (iSupply + threshold)) {
    status = "THEFT: Illegal Load";
    theftActive = true;
  }
  // Sudden Drop
  else if (prevLoad > 1.0 && iLoad < I_OFF) {
    status = "THEFT: Sudden Drop";
    theftActive = true;
  }
  // Reverse Flow
  else if ((iSupply - iLoad) < -REVERSE_THRESH) {
    status = "THEFT: Reverse Flow";
    theftActive = true;
  }

  // ---- Alarm handling ----
  if (theftActive) {
    digitalWrite(ledPin, HIGH);
    digitalWrite(buzzerPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
    digitalWrite(buzzerPin, LOW);
  }

  // ---- Energy calculation (only if NO theft) ----
  if (!theftActive) {
    unsigned long dt_ms = now - lastEnergyMillis;
    if (dt_ms > 0) {
      float P_load = V_NOM * iLoad;
      energy_kWh += (P_load / 1000.0f) * (dt_ms / 3600000.0f);
      lastEnergyMillis = now;
```



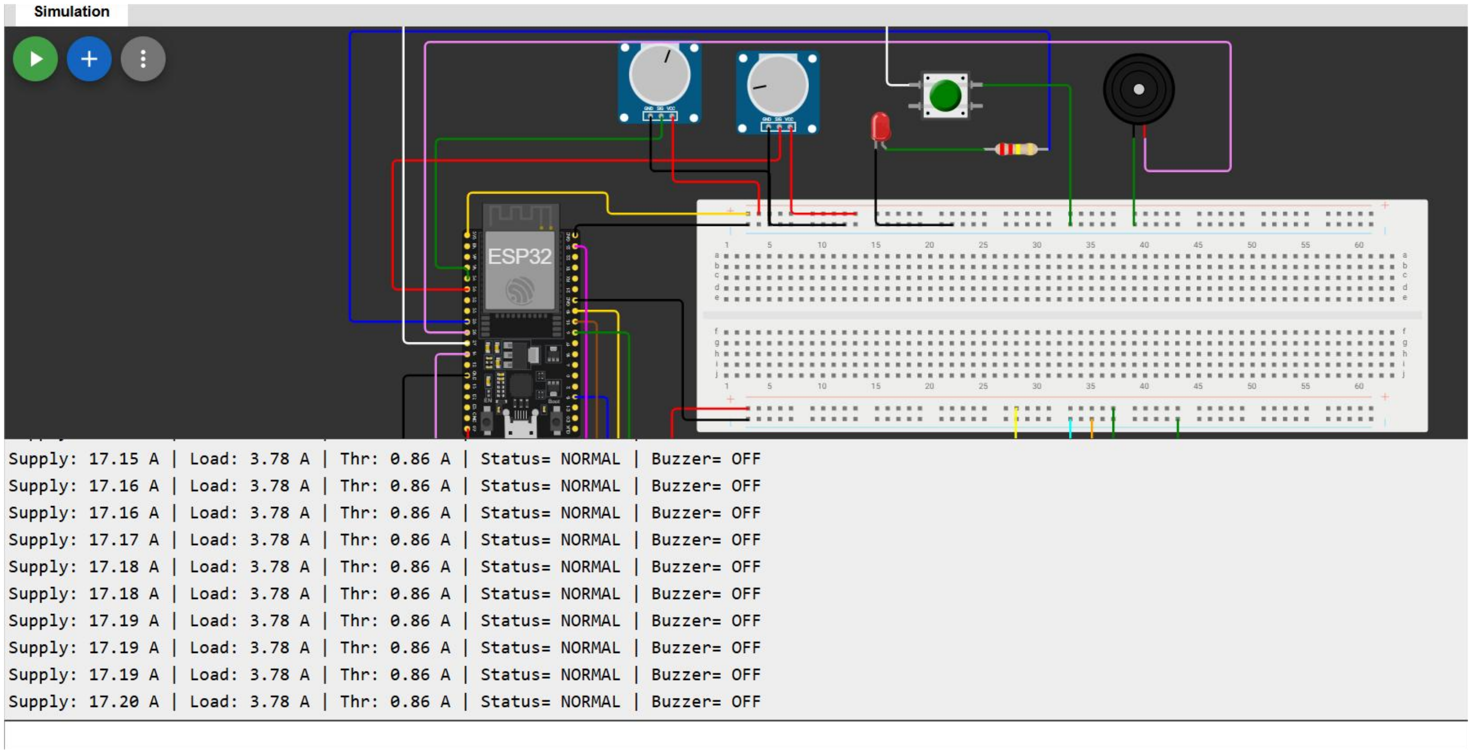
```
void loop() {
  // ---- Energy calculation (only if NO theft) ----
  if (!theftActive) {
    unsigned long dt_ms = now - lastEnergyMillis;
    if (dt_ms > 0) {
      float P_load = V_NOM * iLoad;
      energy_kWh += (P_load / 1000.0f) * (dt_ms / 3600000.0f);
      lastEnergyMillis = now;
    }
  } else {
    // If theft is active, just freeze energy count
    lastEnergyMillis = now;
  }

  // ---- Serial Print ----
  Serial.print("Supply: ");
  Serial.print(iSupply, 2);
  Serial.print(" A | Load: ");
  Serial.print(iLoad, 2);
  Serial.print(" A | Thr: ");
  Serial.print(threshold, 2);
  Serial.print(" A | Status= ");
  Serial.print(status);
  Serial.print(" | Buzzer= ");
  Serial.print(theftActive ? "ON " : "OFF ");
  Serial.println();


  // ---- LCD Update ----
  if (now - lastLcdUpdate >= LCD_UPDATE_MS) {
    lastLcdUpdate = now;
    float cost = energy_kWh * COST_PER_KWH;
    lcd.clear();
    lcd.setCursor(0, 0);
```



```
void loop() {
  // ---- LCD Update ----
  if (now - lastLcdUpdate >= LCD_UPDATE_MS) {
    lastLcdUpdate = now;
    float cost = energy_kWh * COST_PER_KWH;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Units: ");
    lcd.print(energy_kWh, 3);
    lcd.setCursor(0, 1);
    lcd.print("Cost: Rs ");
    lcd.print(cost, 2);
  }

  prevLoad = iLoad;
  delay(250);
}
```

Step 4 - Click on “Start Simulation”

When we will turn the potentiometer we can observe the result in Serial Monitor.



```
Supply: 17.15 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.16 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.16 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.17 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.18 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.18 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.19 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.19 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.19 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
Supply: 17.20 A | Load: 3.78 A | Thr: 0.86 A | Status= NORMAL | Buzzer= OFF
```

Done!!!

Working in Real

We will do the same thing. Just few Differences.

Step 1 - Gather all the required components (ESP32, Breadboard, CT Sensor (30 A), LCD 16×2(I2C), Buzzer, LED, Resistor, Push Button, Jumper wires).

Step 2 - Make the necessary Connections with jumper wire.

Make common 3v3 and GND >> connect 3v3 to +ve rail of horizontal line of breadboard (with F to M Jumper wire) .

GND to -ve rail of breadboard (with F to M Jumper wire).

As we did in Simulator.

Step 3 - We will Connect all the components with ESP32 and Breadboard.

Before that we need to cut the Male Audio Jack, we will get two wires inside , as we are using 2 sensors ,so we need a 4 pin Terminal Block .

Insert both wires of each sensor.

As we can see after that we can easily use jumper wires to connect CT sensor to ESP32.

Solder the I2C with LCD.

Take Two Extension Boards one will be Supply and other will be Load.

Place the phase wire of each extension board in CT sensor properly.

Do the Connection -----

CT Sensors (Current Transformers / Potentiometers in Wokwi)

These act as analog input sensors.

Supply Sensor (CT 1 or Potentiometer 1)

SIG → GPIO 32
VCC → 3.3V Rail of Breadboard
GND → GND Rail of Breadboard

Load Sensor (CT 2 or Potentiometer 2)

SIG → GPIO 35
VCC → 3.3V Rail of Breadboard
GND → GND Rail of Breadboard

2. LED (Indicates Theft / Alarm)

Used to show alert (ON during Theft / Bypass / Attention).

Anode (+) → GPIO 25 (via 220Ω resistor)
Cathode (–) → GND Rail of Breadboard

3. Buzzer
Activates during theft or bypass or attention condition.

Positive (+) → GPIO 26
Negative (–) → GND Rail of Breadboard

4. Push Button (Reset or Manual Control)
Used for manual reset / user interaction (optional).

One leg → GPIO 27
Other leg → GND Rail of Breadboard

5. 16×2 LCD (with I2C module)

VCC → 5V
GND → GND Rail of Breadboard
SDA → GPIO 21
SCL → GPIO 22

6. Power Lines

3.3V rail → For sensors (CTs / potentiometers)
5V rail → For LCD and possibly LED (through

resistor)
GND rail → Common ground for all components.

Step 4 - Now connect ESP32 with Micro USB cable .

Install Arduino IDE Software in Laptop.

Install Important Libraries--
LiquidCrystal I2C
Blynk (To send Data)
ESP32

Open IDE >> Click on Tools >> Boards >> ESP32 >>Select "ESP32 Dev Module" .
Select Tools again >> Ports >> Select the port.

Step 5 - Install Blynk and Setup --

Blynk Setup (Quick Guide)

1. Install Blynk IoT app → Log in

2. Create Template

Name: Smart Energy Meter
Hardware: ESP32
Connection: WiFi
Copy Template ID + Auth Token

3. Add Datastreams (Virtual Pins)
V0 → Supply
V1 → Load
V2 → Difference
V3 → Status

4. Dashboard Widgets

Gauge → V0 (Supply)
Gauge → V1 (Load)
Gauge → V2 (Difference)
Label → V3 (Status)

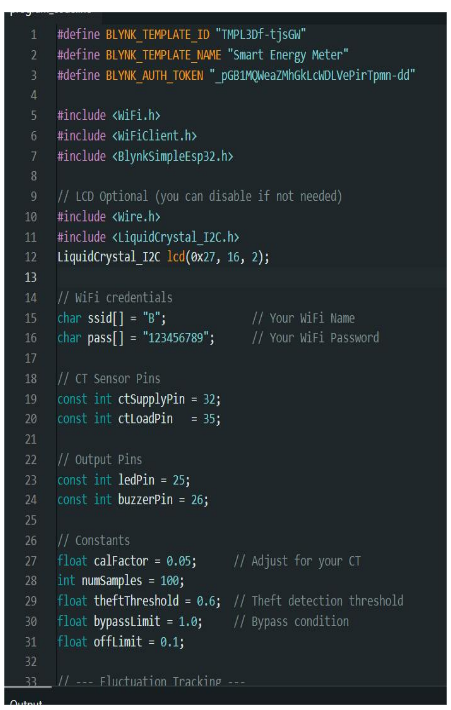
5. Upload Code with:

WiFi name & password
Your Auth Token

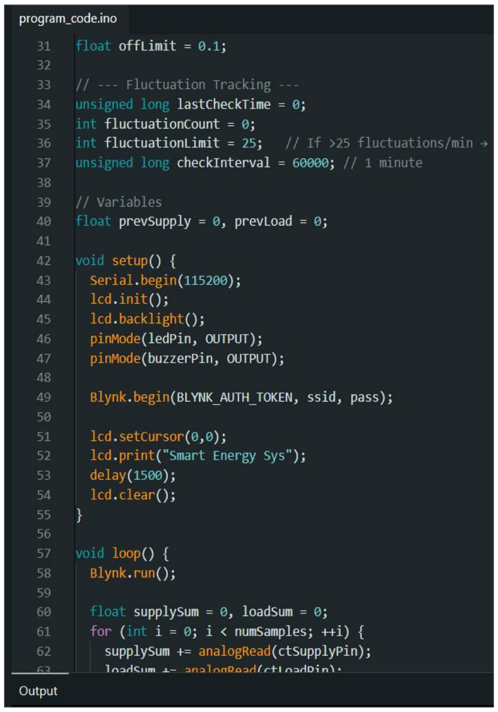
6. Run ESP32 & App

Open Blynk → See live Supply, Load, Status reading.

Step 6 - Upload the code --



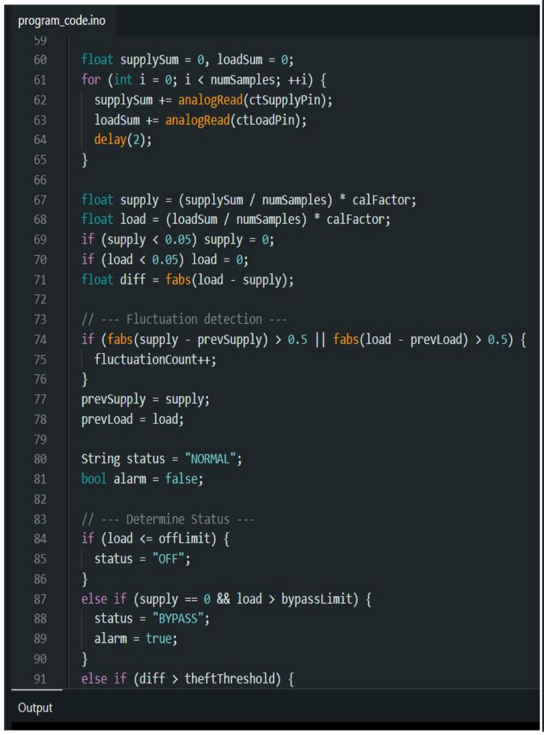
```cpp
#define BLYNK_TEMPLATE_ID "TMPL3Df-tjsGW"
#define BLYNK_TEMPLATE_NAME "Smart Energy Meter"
#define BLYNK_AUTH_TOKEN "_pGB1MQWeaZMhGkLcWDLVePirTpmn-dd"

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

// LCD Optional (you can disable if not needed)
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

// WiFi credentials
char ssid[] = "B";              // Your WiFi Name
char pass[] = "123456789";      // Your WiFi Password

// CT Sensor Pins
const int ctSupplyPin = 32;
const int ctLoadPin   = 35;

// Output Pins
const int ledPin = 25;
const int buzzerPin = 26;

// Constants
float calFactor = 0.05;      // Adjust for your CT
int numSamples = 100;
float theftThreshold = 0.6;  // Theft detection threshold
float bypassLimit = 1.0;     // Bypass condition
float offLimit = 0.1;

// --- Fluctuation Tracking ---
```



program_code.ino

```cpp
float offLimit = 0.1;

// --- Fluctuation Tracking ---
unsigned long lastCheckTime = 0;
int fluctuationCount = 0;
int fluctuationLimit = 25;    // If >25 fluctuations/min →
unsigned long checkInterval = 60000; // 1 minute

// Variables
float prevSupply = 0, prevLoad = 0;

void setup() {
  Serial.begin(115200);
  lcd.init();
  lcd.backlight();
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  lcd.setCursor(0,0);
  lcd.print("Smart Energy Sys");
  delay(1500);
  lcd.clear();
}

void loop() {
  Blynk.run();

  float supplySum = 0, loadSum = 0;
  for (int i = 0; i < numSamples; ++i) {
    supplySum += analogRead(ctSupplyPin);
    loadSum += analogRead(ctLoadPin);
```

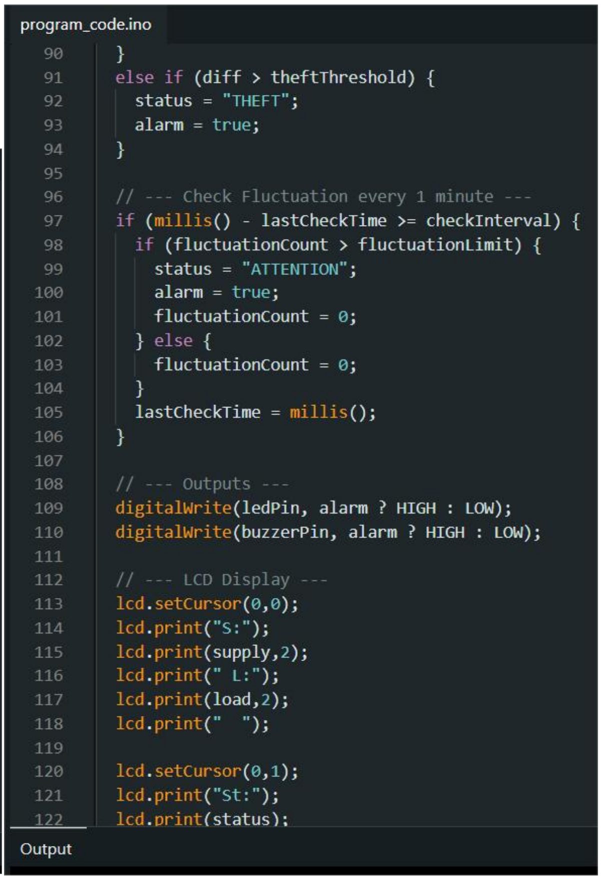
Output



program_code.ino

```cpp
  float supplySum = 0, loadSum = 0;
  for (int i = 0; i < numSamples; ++i) {
    supplySum += analogRead(ctSupplyPin);
    loadSum += analogRead(ctLoadPin);
    delay(2);
  }

  float supply = (supplySum / numSamples) * calFactor;
  float load = (loadSum / numSamples) * calFactor;
  if (supply < 0.05) supply = 0;
  if (load < 0.05) load = 0;
  float diff = fabs(load - supply);

  // --- Fluctuation detection ---
  if (fabs(supply - prevSupply) > 0.5 || fabs(load - prevLoad) > 0.5) {
    fluctuationCount++;
  }
  prevSupply = supply;
  prevLoad = load;

  String status = "NORMAL";
  bool alarm = false;

  // --- Determine Status ---
  if (load <= offLimit) {
    status = "OFF";
  }
  else if (supply == 0 && load > bypassLimit) {
    status = "BYPASS";
    alarm = true;
  }
  else if (diff > theftThreshold) {
```

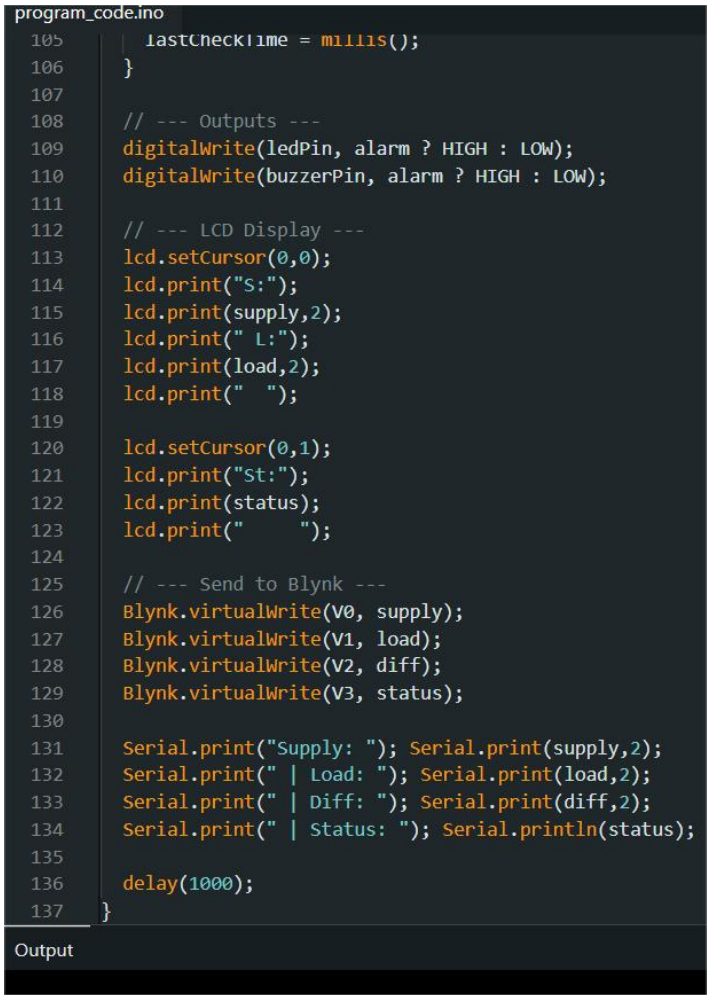
Output



program_code.ino

```cpp
  }
  else if (diff > theftThreshold) {
    status = "THEFT";
    alarm = true;
  }

  // --- Check Fluctuation every 1 minute ---
  if (millis() - lastCheckTime >= checkInterval) {
    if (fluctuationCount > fluctuationLimit) {
      status = "ATTENTION";
      alarm = true;
      fluctuationCount = 0;
    } else {
      fluctuationCount = 0;
    }
    lastCheckTime = millis();
  }

  // --- Outputs ---
  digitalWrite(ledPin, alarm ? HIGH : LOW);
  digitalWrite(buzzerPin, alarm ? HIGH : LOW);

  // --- LCD Display ---
  lcd.setCursor(0,0);
  lcd.print("S:");
  lcd.print(supply,2);
  lcd.print(" L:");
  lcd.print(load,2);
  lcd.print("  ");

  lcd.setCursor(0,1);
  lcd.print("St:");
  lcd.print(status);
```

Output



program_code.ino

```
    lastCheckTime = millis();
  }

  // --- Outputs ---
  digitalWrite(ledPin, alarm ? HIGH : LOW);
  digitalWrite(buzzerPin, alarm ? HIGH : LOW);

  // --- LCD Display ---
  lcd.setCursor(0,0);
  lcd.print("S:");
  lcd.print(supply,2);
  lcd.print(" L:");
  lcd.print(load,2);
  lcd.print("  ");

  lcd.setCursor(0,1);
  lcd.print("St:");
  lcd.print(status);
  lcd.print("     ");

  // --- Send to Blynk ---
  Blynk.virtualWrite(V0, supply);
  Blynk.virtualWrite(V1, load);
  Blynk.virtualWrite(V2, diff);
  Blynk.virtualWrite(V3, status);

  Serial.print("Supply: "); Serial.print(supply,2);
  Serial.print(" | Load: "); Serial.print(load,2);
  Serial.print(" | Diff: "); Serial.print(diff,2);
  Serial.print(" | Status: "); Serial.println(status);

  delay(1000);
}
```

Output

Readings will be visible in Blynk.

Done !!!