

# **BABU BANARSI DAS**

## **University**



## **Internet of Things Project File**

### **(NITBC4351)**

**SUBMITTED BY:**

**Adarsh Kumar | Abhinav Singh  
|Netraja | Sakshi Choubey  
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).

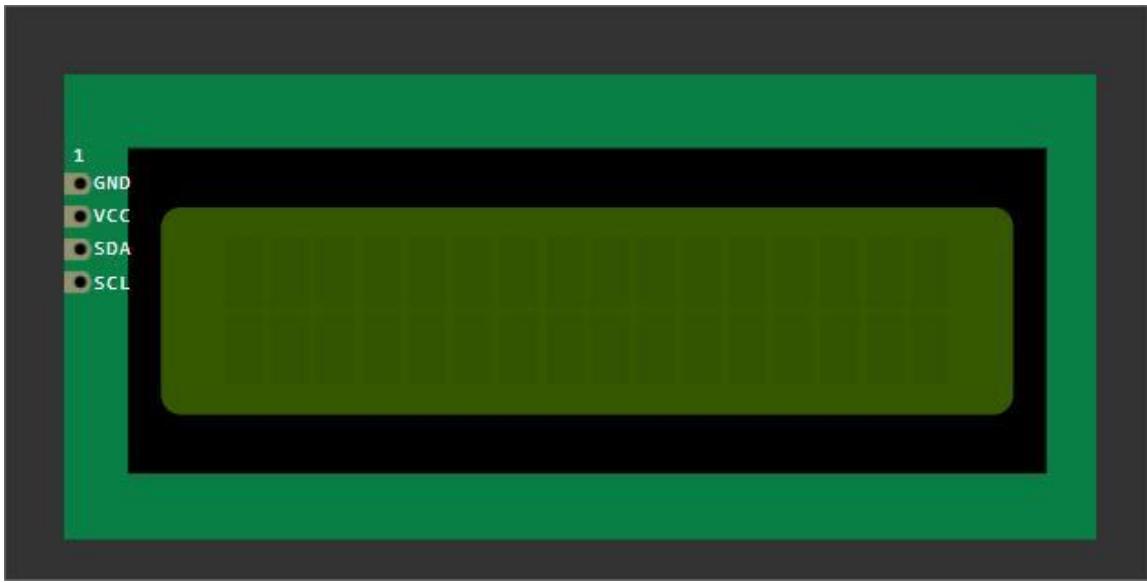
8. SDA/SCL Pins (SDA=GPIO21, SCL=GPIO22): Used for I<sup>2</sup>C communication.

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 (I<sup>2</sup>C) : A 16×2 LCD with I<sup>2</sup>C displays two lines of text (16 characters each) and communicates using only two data wires for easier connections.**



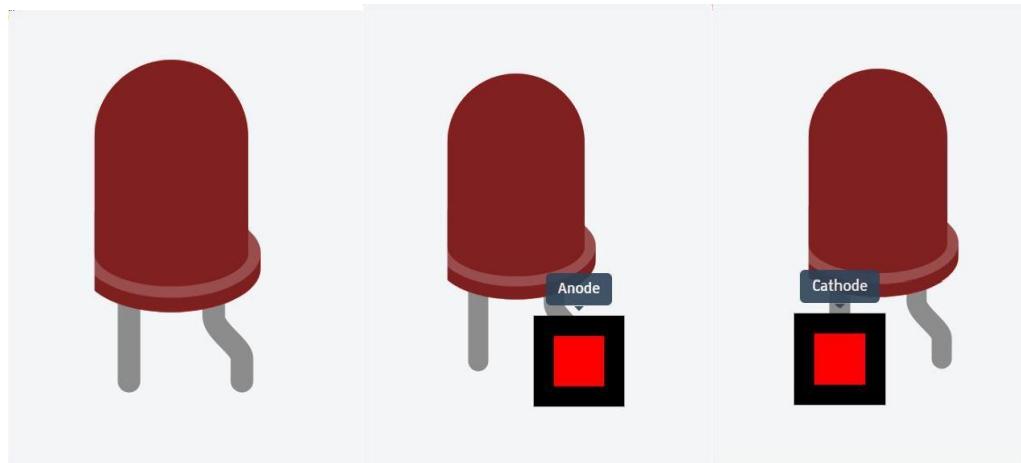
VCC: Supplies power to the LCD module.

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.**

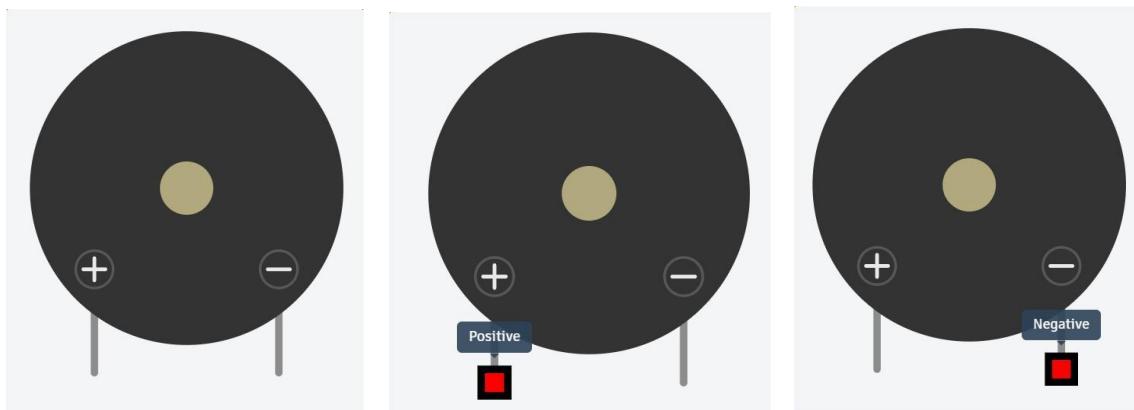


**Anode (+): The longer leg of the LED. It is the positive terminal where**

current enters.

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

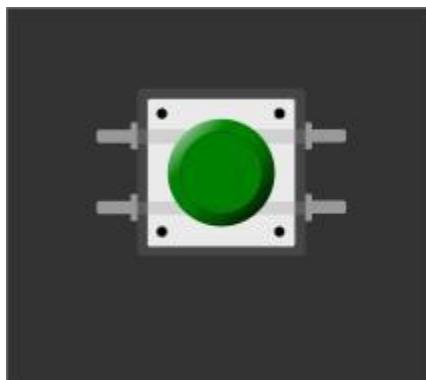
**Tool 5: Buzzer :** A buzzer converts electrical signals into sound to provide audible alerts or indications.



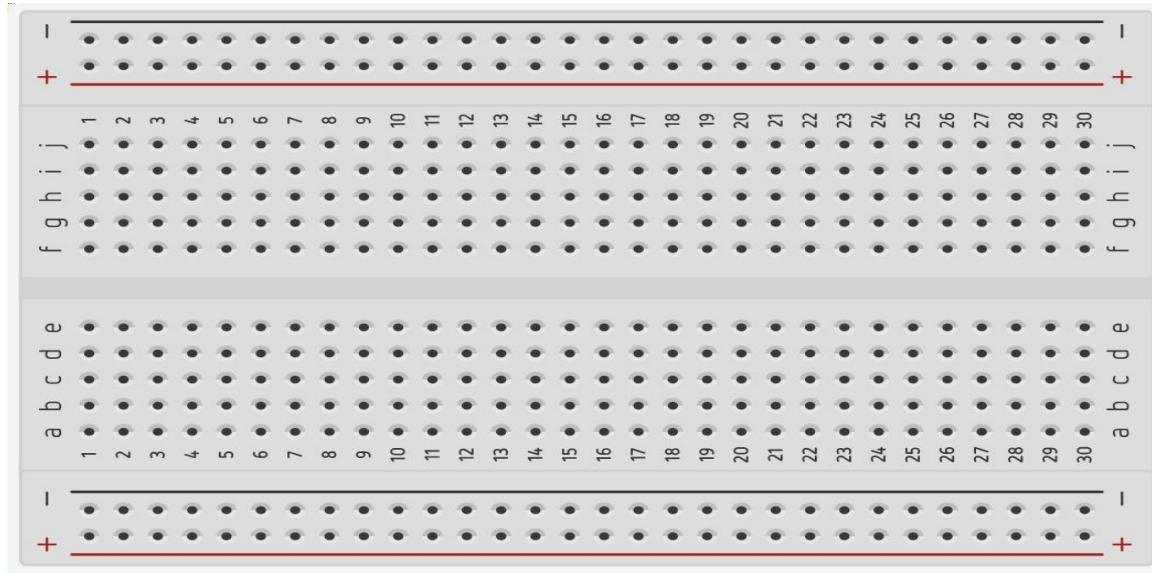
Positive pin: Connects to the power supply to activate the buzzer.

Negative pin: Connects to ground to complete the circuit.

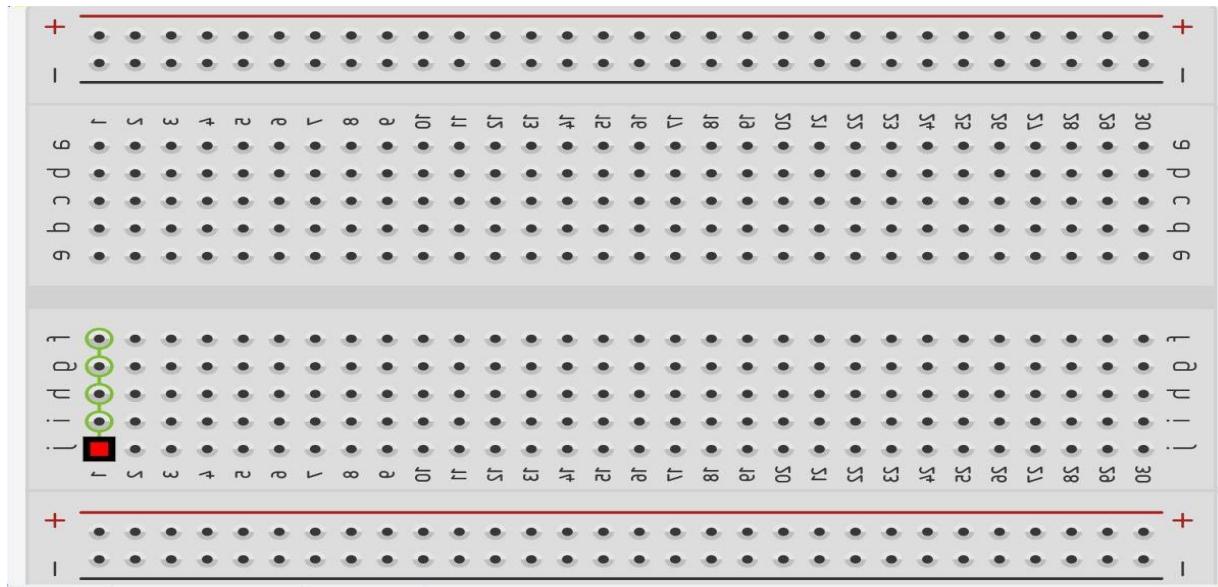
**Tool 6 : Push Button :** A push button is a simple switch that connects or disconnects a circuit when pressed.



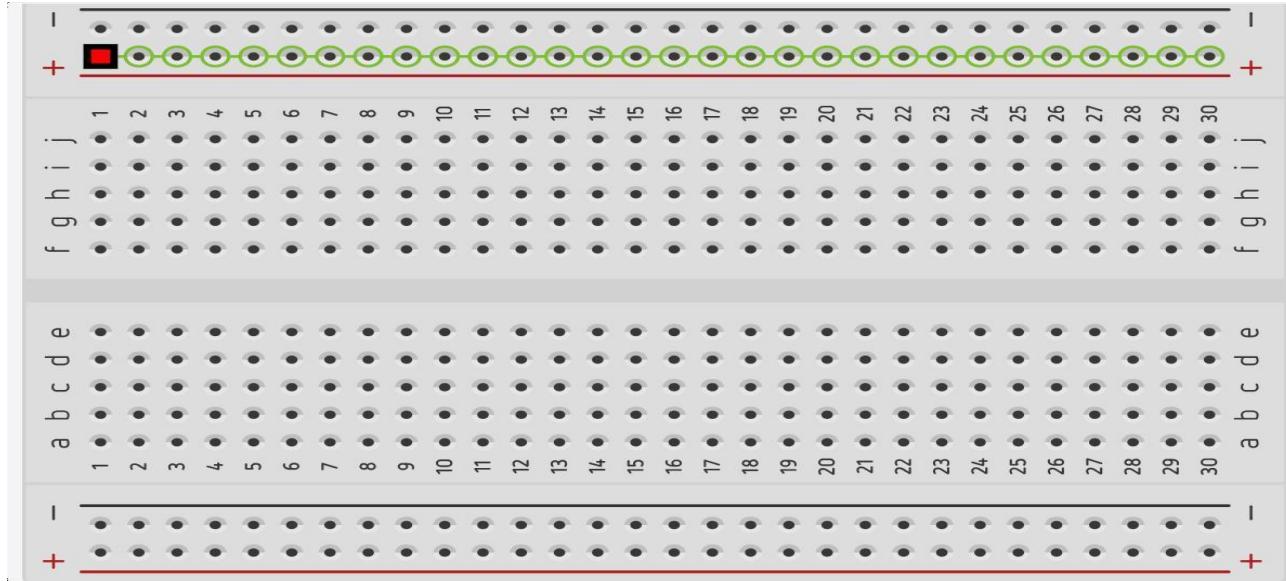
**Tool 7: Breadboard:** A board used to connect electronic components without soldering.



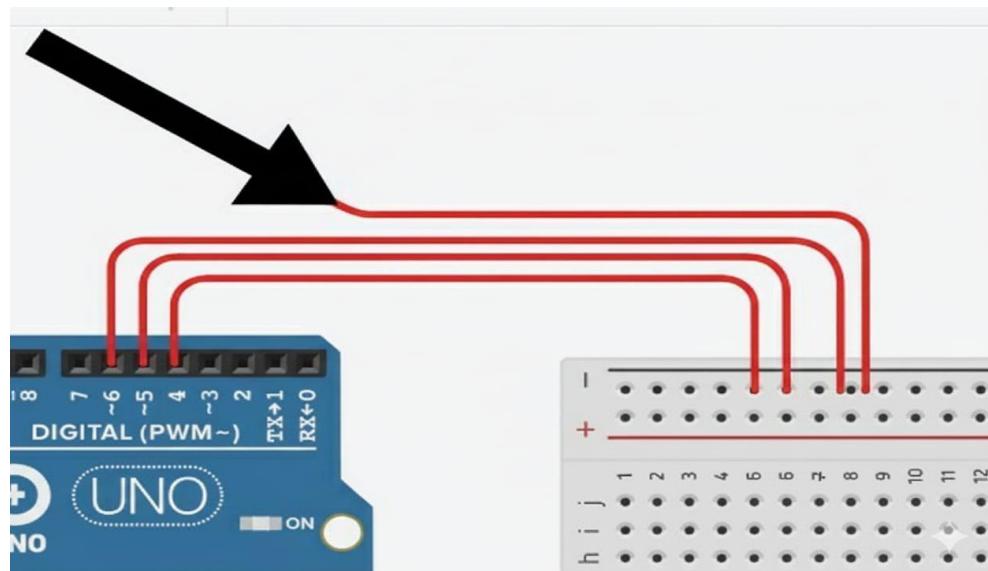
**Vertical Lines:** Side rails for power supply (+ and -).



**Horizontal Lines:** Middle rows where components are connected.

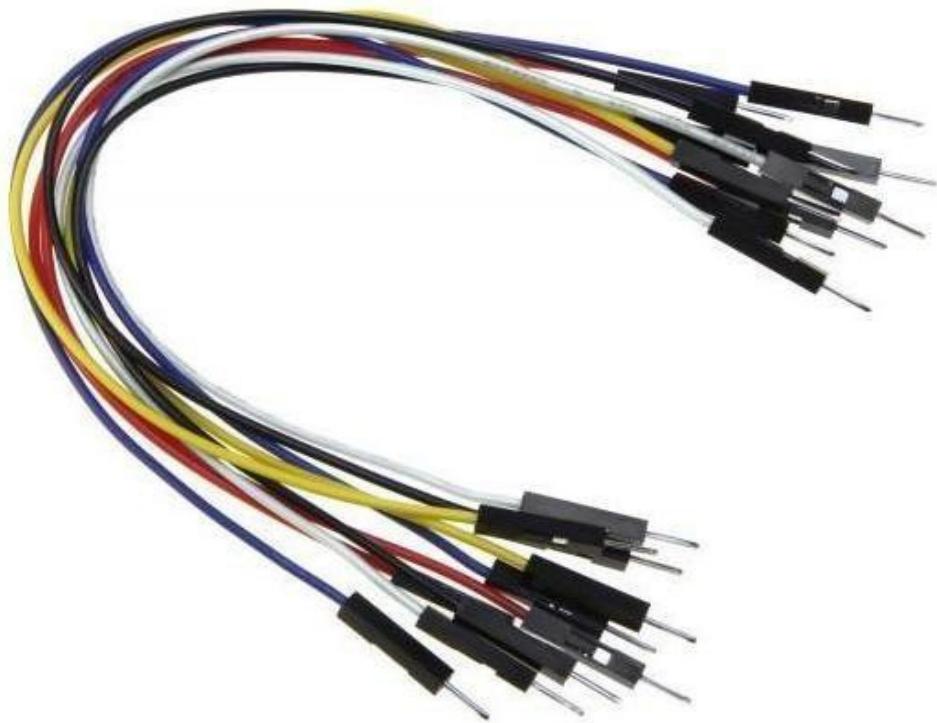


**Tool 8: Jumper Wires : Jumper Wires:** Small connecting wires used to make connections between components and the breadboard/Arduino.

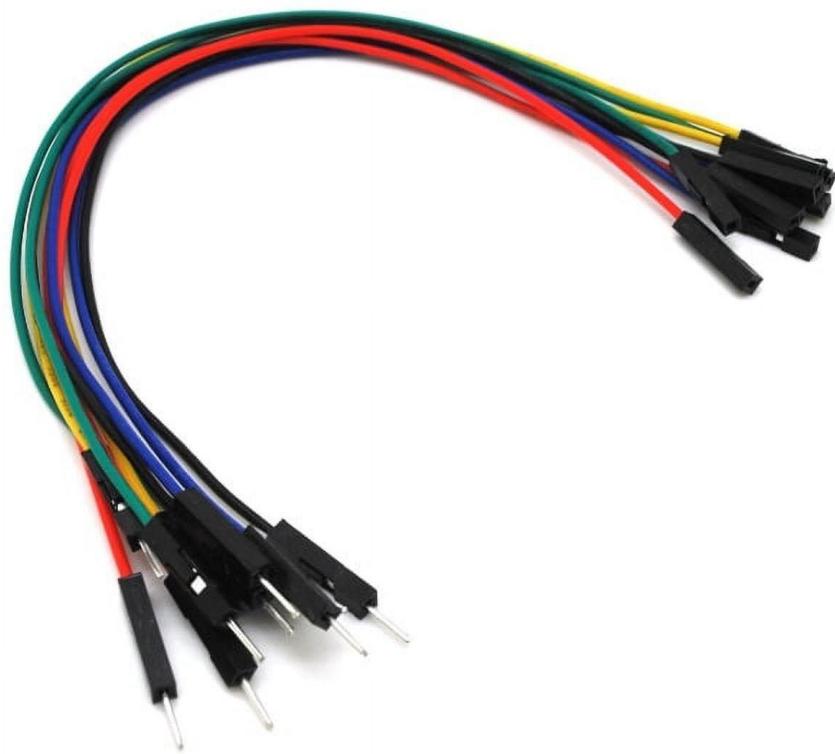


**Types of Jumping Wires:**

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



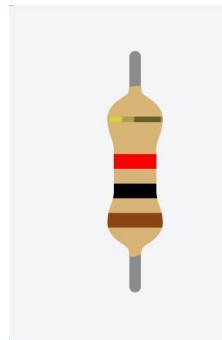
**2- Male to Female Jumper Wires:** Used to connect Arduino pins (male) to components with female headers



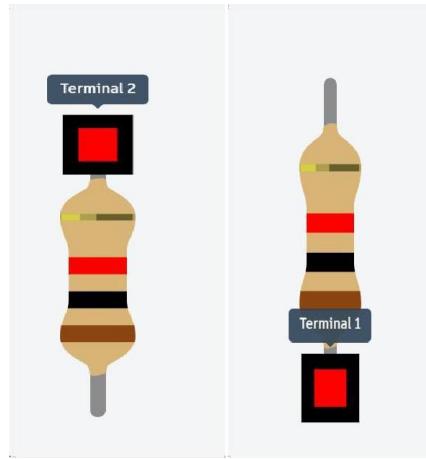
**3- Female to Female Jumper Wires:** Used to connect two components or modules that both have male pins.



**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.



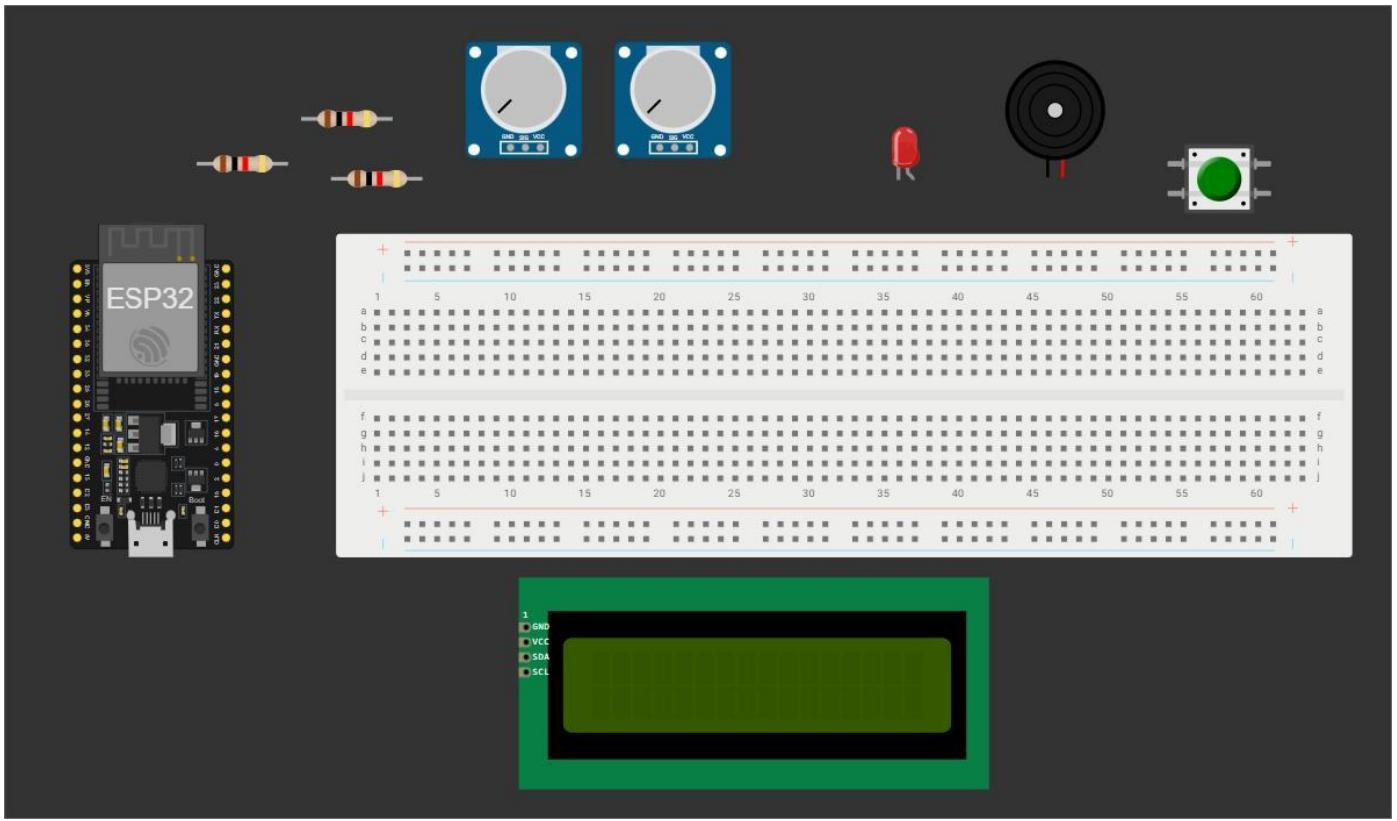
**Tool 10 : Extension Boards :** An extension board provides multiple power outlets from a single socket to connect several devices at once.



## Working (in Wokwi Simulator ) :

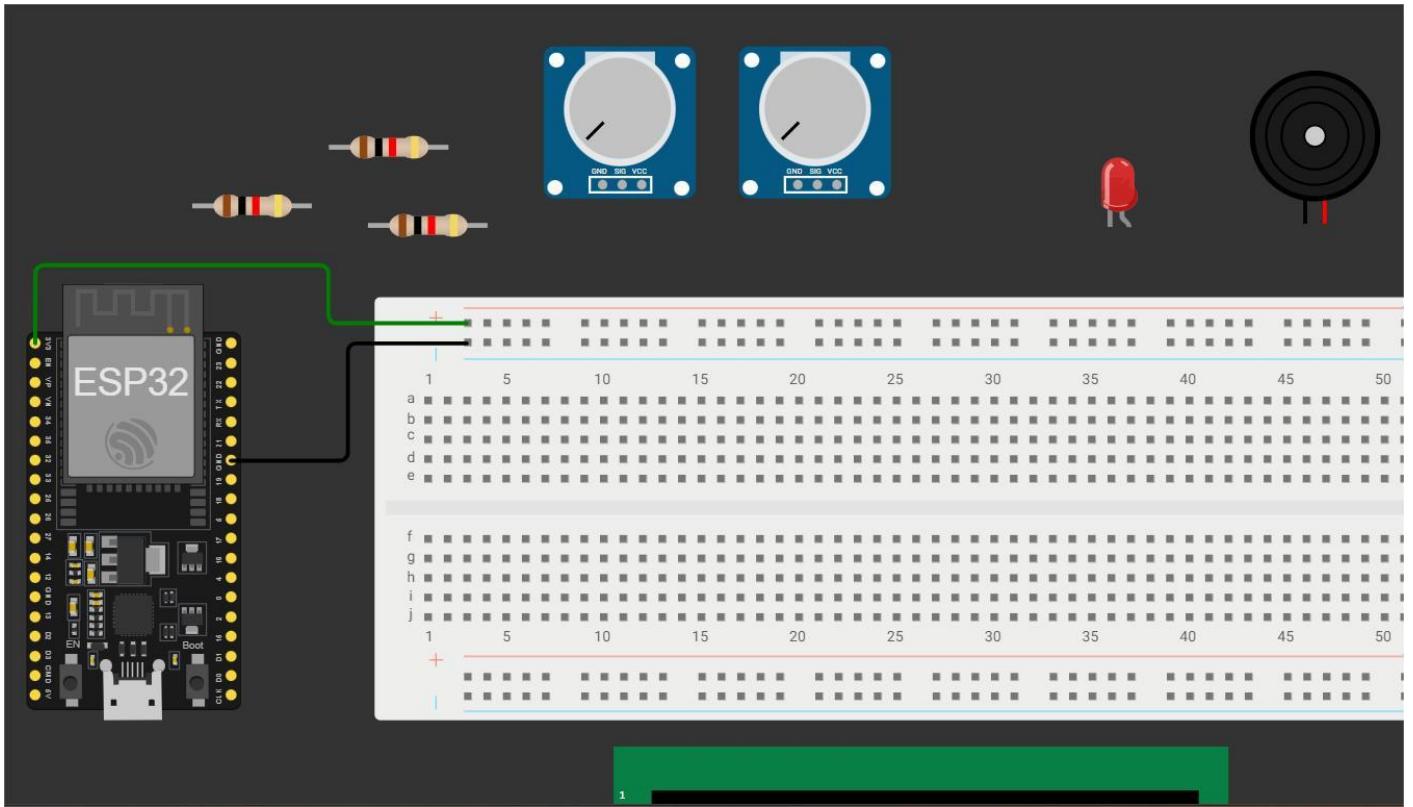
Note : Since Wokwi doesn't provide CT Sensors so, we will use Potentiometer to Demonstrate the project.

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 I<sup>2</sup>C)

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

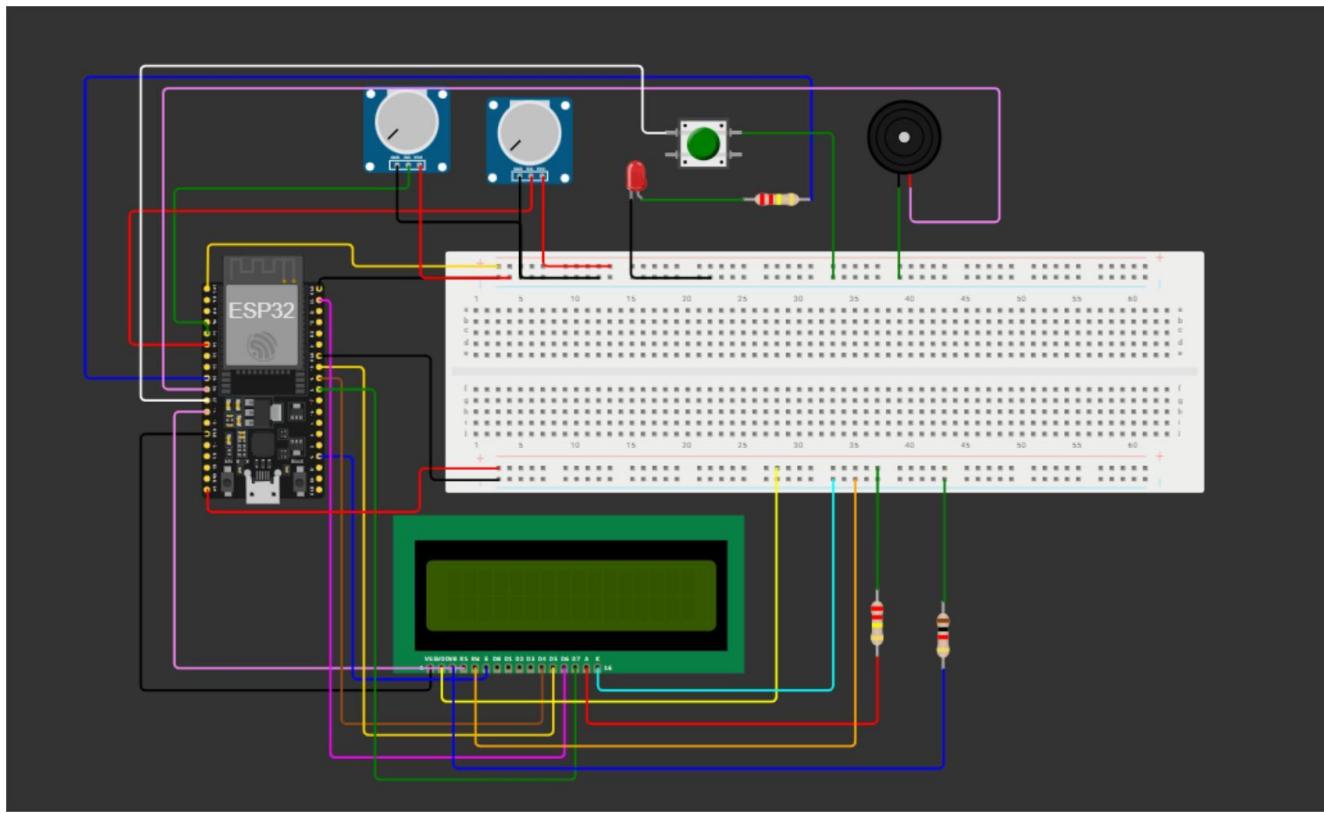
K (Backlight -) → GND rail

#### 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 -

```

    potSupplyPin = 3A;
    p0tLoadP1n   = 35j
    ledPin        = 25;
    buzzerPTn    = 2Ej

LiquidCrysTaT lco 14, 15, 18, 19, 23, 5;

V_NOM = 230.0;
COS*_PER_KWH 7.5;

16      I OFF = 0.1;

I MIN THEFT = 0.5;
*HEFT_PCT = 0.05;
REVERSE_THRESH = 0.1;

LCD_UPDATE_MS = 1000;

isupply = 0.0, iLoad = 0.0;
prevLoad = 0.0;
energy kWh 0.0;

lastEnergyN1l11s = 8;
lastLcdUpdate = 0;

theftAct1ve =     ;

```

```

31 // Theft flag
32 bool theftActive = false;
33
34 // smoothing
35 const float EMA_ALPHA = 0.12;
36
37 // helper: ADC raw to Amps
38 float rawToAmp(int raw) {
39     return (raw / 4095.0f) * I_MAX;
40 }
41
42 void setup() {
43     Serial.begin(115200);
44     pinMode(ledPin, OUTPUT);
45     pinMode(buzzerPin, OUTPUT);
46
47     digitalWrite(ledPin, LOW);
48     digitalWrite(buzzerPin, LOW);
49
50     lcd.begin(16, 2);
51     lcd.print("System Ready");
52     delay(1000);
53     lcd.clear();
54
55     lastEnergyMillis = millis();
56     lastLcdUpdate = millis();
57 }
58
59 void loop() {
60     unsigned long now = millis();
61
62     // ---- Read pots ----
63     int rawSupply = analogRead(potSupplyPin);
64

```

```

60 void loop() {
61
62 }
```

```

rauSup1y = analogRead(potSupplyPin);
rauLoad = analogRead(potLoadPin);
if(measuredSupply rauToAcp(rauSupply));
measuredLoad = raw oAcp(rauLoad);
```

```

60 if (iLoad > isupply + treshold
status = "THEFT Illegal Load";
theftActive =
```

OPtvLoad > 1,8 f8 iLoad < I OFF

```

7s    ii (iSup1y 0.0f) iSupTy measuredSupply,
1 else isupply = ENA ALPHA " measuredSupply + (1 - ENA ALPHA) * isupply;
```

```

:iLoad == 8,8^"iLoad = measuredLoad;
iLoad = Ed ALPKA * measuredLoad + [ 1 - EMA_ALPHA ] * itoad;
isupply = iSupply - iLoad / -REVERSE_THRESH
status =
theftActive =
```

threshold = | HC\_TOfeFT, iSupply " \*HEFT\_PCT .

```

status =
theftActive =
```

```

'itoad < I OFF
```

```

theftActive
leoPir,
buzzerPin,
```

```

leoPin,
buzzerIn,
```

```

iSupDiy <= I OFF && iLoad >= I ON
status = ""<L
theftActive
```

```

!theftActive) {
signed long dt_ms = now - lastEnergyN11s;
dt_ms > 0
```

```

se II (iLoad > isupply + threshold
status = "Hfif-1: Illegal Load";
theftActive = ->'us:
```

```

P load = V U0f1 ' iLoad;
energy kin += {P load / 1600.0f1 * (dt ns / J600000.0f1);
```

```

60   void loop() {
117     // ---- Energy calculation (only if NO theft) ----
118     if (!theftActive) {
119       unsigned long dt_ms = now - lastEnergyMillis;
120       if (dt_ms > 0) {
121         float P_load = V_NOM * iLoad;
122         energy_kwh += (P_load / 1000.0f) * (dt_ms / 3600000.0f);
123         lastEnergyMillis = now;
124       }
125     } else {
126       // If theft is active, just freeze energy count
127       lastEnergyMillis = now;
128     }
129
130     // ---- Serial Print ----
131     Serial.print("Supply: ");
132     Serial.print(isupply, 2);
133     Serial.print(" A | Load: ");
134     Serial.print(iLoad, 2);
135     Serial.print(" A | Thr: ");
136     Serial.print(threshold, 2);
137     Serial.print(" A | Status= ");
138     Serial.print(status);
139     Serial.print(" | Buzzer= ");
140     Serial.print(theftActive ? "ON" : "OFF");
141     Serial.println();
142
143
144     // ---- LCD Update ----
145     if (now - lastLcdUpdate >= LCD_UPDATE_MS) {
146       lastLcdUpdate = now;
147       float cost = energy_kwh * COST_PER_KWH;
148       lcd.clear();
149       lcd.setCursor(0, 0);

```

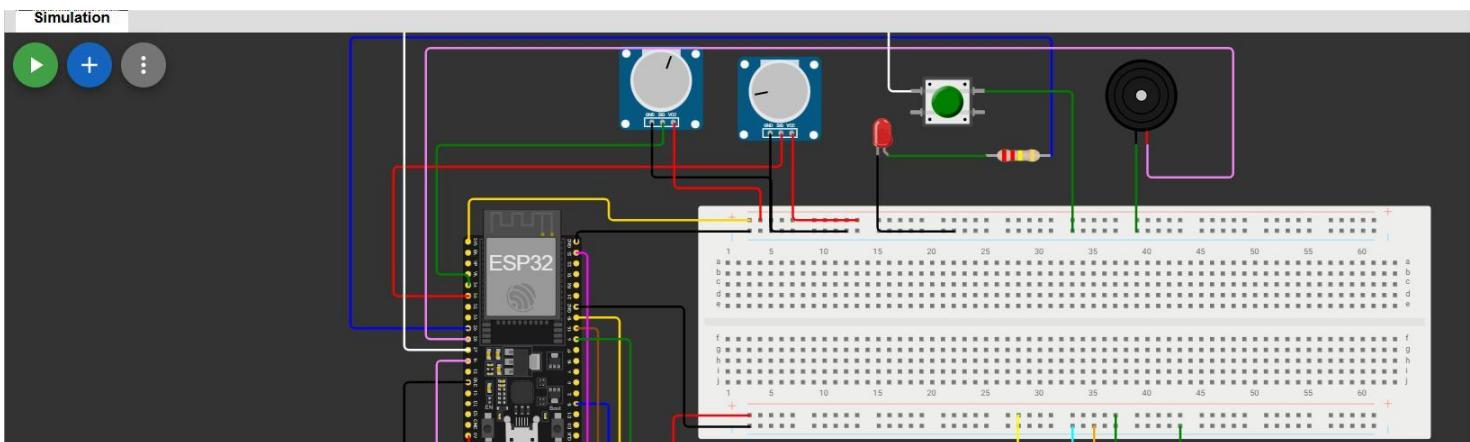
```

60   void loop() {
144     // ---- LCD Update ----
145     if (now - lastLcdUpdate >= LCD_UPDATE_MS) {
146       lastLcdUpdate = now;
147       float cost = energy_kwh * COST_PER_KWH;
148       lcd.clear();
149       lcd.setCursor(0, 0);
150       lcd.print("Units: ");
151       lcd.print(energy_kwh, 3);
152       lcd.setCursor(0, 1);
153       lcd.print("Cost: Rs ");
154       lcd.print(cost, 2);
155     }
156
157     prevLoad = iLoad;
158     delay(250);
159   }
160

```

#### 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  $220\Omega$  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. $16 \times 2$ LCD (with I<sub>2</sub>C 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 --

```

1 #define BLYNK_TEMPLATE_ID "TMPL3DF-tj5m"
2 #define BLYNK_TEMPLATE_NAME "Smart Energy Meter"
3 #define BLYNK_AUTH_TOKEN "pGB1M0WeaZMhgkLcwDLVePirTpmm-dd"
4
5 #include <WiFi.h>
6 #include <WiFiClient.h>
7 #include <BlynkSimpleEsp32.h>
8
9 // LCD Optional (you can disable if not needed)
10 #include <Wire.h>
11 #include <LiquidCrystal_I2C.h>
12 LiquidCrystal_I2C lcd(0x27, 16, 2);
13
14 // WiFi credentials
15 char ssid[] = "B";           // Your WiFi Name
16 char pass[] = "123456789";   // Your WiFi Password
17
18 // CT Sensor Pins
19 const int ctSupplyPin = 32;
20 const int ctLoadPin = 35;
21
22 // Output Pins
23 const int ledPin = 25;
24 const int buzzerPin = 26;
25
26 // Constants
27 float calFactor = 0.05;      // Adjust for your CT
28 int numSamples = 100;
29 float theftThreshold = 0.6; // Theft detection threshold
30 float bypassLimit = 1.0;    // Bypass condition
31 float offlimit = 0.1;
32
33 // --- Fluctuation Tracking ...

```

Output

```

program_code.ino
31 float offlimit = 0.1;
32
33 // --- Fluctuation Tracking ---
34 unsigned long lastCheckTime = 0;
35 int fluctuationCount = 0;
36 int fluctuationLimit = 25; // If >25 fluctuations/min →
37 unsigned long checkInterval = 60000; // 1 minute
38
39 // Variables
40 float prevSupply = 0, prevLoad = 0;
41
42 void setup() {
43   Serial.begin(115200);
44   lcd.init();
45   lcd.backlight();
46   pinMode(ledPin, OUTPUT);
47   pinMode(buzzerPin, OUTPUT);
48
49   Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
50
51   lcd.setCursor(0,0);
52   lcd.print("Smart Energy sys");
53   delay(1500);
54   lcd.clear();
55 }
56
57 void loop() {
58   Blynk.run();
59
60   float supplySum = 0, loadSum = 0;
61   for (int i = 0; i < numSamples; ++i) {
62     supplySum += analogRead(ctSupplyPin);
63     loadSum += analogRead(ctLoadPin);
64     delay(2);
65   }
66
67   float supply = (supplySum / numSamples) * calFactor;
68   float load = (loadSum / numSamples) * calFactor;
69   if (supply < 0.05) supply = 0;
70   if (load < 0.05) load = 0;
71   float diff = fabs(load - supply);
72
73   // --- Fluctuation detection ---
74   if (fabs(supply - prevSupply) > 0.5 || fabs(load - prevLoad) > 0.5) {
75     fluctuationCount++;
76   }
77   prevSupply = supply;
78   prevLoad = load;
79
80   String status = "NORMAL";
81   bool alarm = false;
82
83   // --- Determine Status ---
84   if (load <= offlimit) {
85     status = "OFF";
86   }
87   else if (supply == 0 && load > bypassLimit) {
88     status = "BYPASS";
89     alarm = true;
90   }
91   else if (diff > theftThreshold) {

```

Output

```

program_code.ino
59
60   float supplySum = 0, loadSum = 0;
61   for (int i = 0; i < numSamples; ++i) {
62     supplySum += analogRead(ctSupplyPin);
63     loadSum += analogRead(ctLoadPin);
64     delay(2);
65   }
66
67   float supply = (supplySum / numSamples) * calFactor;
68   float load = (loadSum / numSamples) * calFactor;
69   if (supply < 0.05) supply = 0;
70   if (load < 0.05) load = 0;
71   float diff = fabs(load - supply);
72
73   // --- Fluctuation detection ---
74   if (fabs(supply - prevSupply) > 0.5 || fabs(load - prevLoad) > 0.5) {
75     fluctuationCount++;
76   }
77   prevSupply = supply;
78   prevLoad = load;
79
80   String status = "NORMAL";
81   bool alarm = false;
82
83   // --- Determine Status ---
84   if (load <= offlimit) {
85     status = "OFF";
86   }
87   else if (supply == 0 && load > bypassLimit) {
88     status = "BYPASS";
89     alarm = true;
90   }
91   else if (diff > theftThreshold) {

```

Output

```

program_code.ino
90   }
91   else if (diff > theftThreshold) {
92     status = "THEFT";
93     alarm = true;
94   }
95
96   // --- Check Fluctuation every 1 minute ---
97   if (millis() - lastCheckTime >= checkInterval) {
98     if (fluctuationCount > fluctuationLimit) {
99       status = "ATTENTION";
100      alarm = true;
101      fluctuationCount = 0;
102    } else {
103      fluctuationCount = 0;
104    }
105    lastCheckTime = millis();
106  }
107
108  // --- Outputs ---
109  digitalWrite(ledPin, alarm ? HIGH : LOW);
110  digitalWrite(buzzerPin, alarm ? HIGH : LOW);
111
112  // --- LCD Display ---
113  lcd.setCursor(0,0);
114  lcd.print("S:");
115  lcd.print(supply,2);
116  lcd.print(" L:");
117  lcd.print(load,2);
118  lcd.print(" ");
119
120  lcd.setCursor(0,1);
121  lcd.print("St:");
122  lcd.print(status);

```

Output

```
program_code.ino
105     |     lastCheckTime = millis();
106 }
107
108 // --- Outputs ---
109 digitalWrite(ledPin, alarm ? HIGH : LOW);
110 digitalWrite(buzzerPin, alarm ? HIGH : LOW);
111
112 // --- LCD Display ---
113 lcd.setCursor(0,0);
114 lcd.print("S:");
115 lcd.print(supply,2);
116 lcd.print(" L:");
117 lcd.print(load,2);
118 lcd.print("   ");
119
120 lcd.setCursor(0,1);
121 lcd.print("St:");
122 lcd.print(status);
123 lcd.print("   ");
124
125 // --- Send to Blynk ---
126 Blynk.virtualWrite(V0, supply);
127 Blynk.virtualWrite(V1, load);
128 Blynk.virtualWrite(V2, diff);
129 Blynk.virtualWrite(V3, status);
130
131 Serial.print("Supply: "); Serial.print(supply,2);
132 Serial.print(" | Load: "); Serial.print(load,2);
133 Serial.print(" | Diff: "); Serial.print(diff,2);
134 Serial.print(" | Status: "); Serial.println(status);
135
136 delay(1000);
137 }

Output
```

Readings will be visible in Blynk.

Done !!!