Slab Guardian: Full Wiring Story, Breadboard Position by Position

Read this in order, nothing skipped, from empty breadboard to live bulb data on your phone.

Understanding your breadboard first (30 seconds, matters a lot)

Look at your breadboard. Top and bottom edges have two long rows each, usually marked with a red line (+) and a blue/black line (-), these are your power rails, every hole along a red line is electrically connected to every other hole on that same red line, same for blue.

The main middle area is split into two halves by a center gap/notch. In each half, holes are grouped in short vertical columns of 5 (labeled a-e on one side, f-j on the other), each column of 5 holes is electrically connected to each other vertically, but NOT connected across the center gap, and NOT connected to the column next to it.

That's the whole logic of a breadboard: rails run horizontally the full length, columns run vertically in short groups of 5, center gap breaks the connection between the two halves.

Step 1: Place the ESP32

Lay your breadboard flat, power rails running left to right at top and bottom. Take your ESP32 and straddle it directly across the center gap, so one row of its pins sits in the column holes on the left half, and the other row of pins sits in the column holes on the right half. Push it in firmly near one end of the board, leaving the rest of the breadboard free for PZEM and jumpers.

Each individual ESP32 pin now owns one small column of 5 holes on its side. That column is now "that pin," electrically, anywhere else in that same column of 5 is the same electrical point as the pin itself.

Step 2: Bring power to the rails

Take one male to male jumper wire. Put one end into the same column as your ESP32's 5V pin (any of the remaining 4 holes in that column works, they're all identical). Put the other end into the red (+) rail.

Take a second male to male jumper wire. Put one end into the same column as your ESP32's GND pin. Put the other end into the blue/black (-) rail.

You now have your ESP32's power available anywhere along those two rails, this is the whole point of using the rails, so you don't need to run separate wires all the way back to the chip's actual pins for every single component.

Step 3: Bring PZEM into the picture

Place your PZEM board flat on the breadboard area past the ESP32, or just beside it on the table if it doesn't have pins that insert into a breadboard (many PZEM-004T boards have a pin header you plug male to female jumpers into directly rather than inserting the board itself). Check your specific board, if it has a 4 pin male header sticking up, you'll be plugging female jumper ends onto those pins directly, not inserting it into breadboard holes.

Step 4: Wire PZEM's power

Take a male to female jumper wire. Female end onto PZEM's **5V** pin. Male end into the red (+) rail on the breadboard, anywhere along that rail.

Take another male to female jumper wire. Female end onto PZEM's **GND** pin. Male end into the blue/black (-) rail, anywhere along it.

PZEM is now powered from the same source as your ESP32.

Step 5: Wire the data lines, this is the part people get backwards

Take a male to female jumper wire. Female end onto PZEM's **TX** pin. Male end goes directly into the same column as ESP32's **RX2 pin (GPIO16)**, not the rail, this is a direct point to point signal connection.

Take another male to female jumper wire. Female end onto PZEM's **RX** pin. Male end goes directly into the same column as ESP32's **TX2 pin (GPIO17)**.

Say it out loud while you do it: PZEM TX to ESP32 RX2, PZEM RX to ESP32 TX2. If you connect TX to TX, nothing will ever work, this is the single most common mistake at this stage.

Step 6: Plug in the clamp

Take your SCT-013 with the soldered 3.5mm jack. Push it firmly into PZEM's CT socket until it seats.

Step 7: Visual double check before power

Look at your full breadboard right now. You should see: ESP32 straddling the center gap, two jumpers running from its 5V and GND columns to the two rails, PZEM connected by 4 jumpers total, two to the rails for power, two direct point to point for TX/RX, and the SCT jack plugged into PZEM. Nothing yet is connected to mains. Nothing should be loose or half inserted, push every jumper in firmly, a loose jumper causes exactly the kind of random failure that wastes an hour of guessing.

Step 8: Power it up over USB and test

Plug your USB-C cable from ESP32 into your laptop. Open Arduino IDE, upload a sketch that initializes Serial2 on RX2/TX2, creates a PZEM object on that serial, and every 1-2 seconds reads and prints voltage, current, power, and energy to Serial Monitor.

Open Serial Monitor. You should see numbers printing, likely zeros or error values right now since nothing is connected to mains yet, that's expected, you're only confirming the wiring/communication itself works, not real readings yet.

If you see garbage text or nothing at all, the TX/RX pair is probably swapped, recheck Step 5.

Step 9: Set up the bulb circuit, mains involved from here

Turn off the relevant breaker before touching anything below. Take an extension cord or lamp holder with your bulb. With it unplugged from the wall, connect the mains live wire into PZEM's L (line) input terminal, and back out through L output to the bulb. Connect neutral straight through as well, into PZEM's N terminal and onward to the bulb.

Step 10: Clamp placement

Open the SCT-013 clamp and close it around the single live wire feeding the bulb, the same wire going into PZEM's L input. Never clamp both live and neutral together, never clamp only neutral, either mistake reads close to zero.

Step 11: Power on and verify

Plug the extension cord into the wall. Your ESP32 should still be running off your laptop's USB from Step 8. Watch Serial Monitor, you should now see roughly 220V showing up. Turn the bulb on, watch power/current rise. Turn it off, watch them drop back toward zero. This confirms your whole sensing chain works end to end.

Step 12: Add WiFi and MQTT

Add WiFi.begin() with your network's SSID and password to your sketch. Install the "PubSubClient" library. Add code to connect to broker `test.mosquitto.org` on port 1883. Every 2-3 seconds, build a small JSON string with your current readings and publish it to a topic you make up yourself, something unique so you don't collide with strangers also using this public broker, for example `slabguardian/yourname123/data`.

Step 13: See it live on your phone

Install "MQTT Explorer" or "IoT MQTT Panel" from your phone's app store, both are free. Open it, connect to `test.mosquitto.org` on port 1883, no username or password needed for this public test broker. Subscribe to the exact same topic string you used in your firmware.

Turn the bulb on and off while watching your phone. The numbers should update live, wirelessly, sourced from your actual physical sensor. That's your full chain complete, wire on breadboard to live number on a screen.