

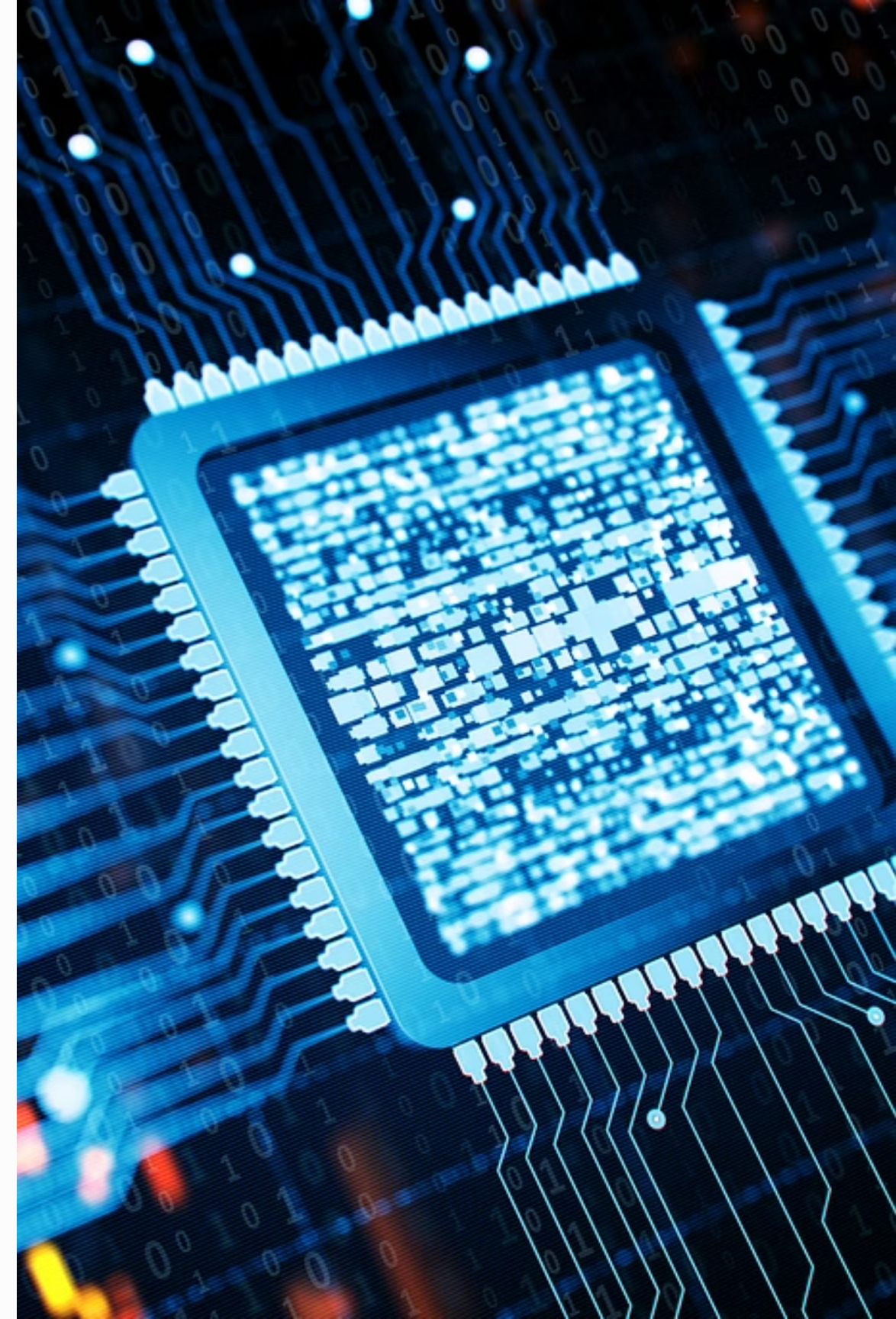
From Hardware to hub: Custom Firmware & Communication

Taking control of your devices and understanding the languages they speak.

devstaff

Michalis Raptakis

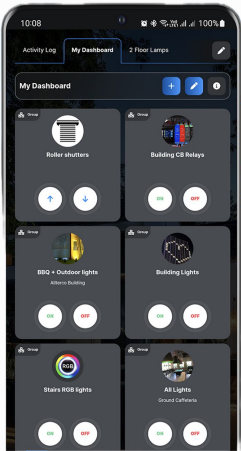
Backend Engineer @Fairlo



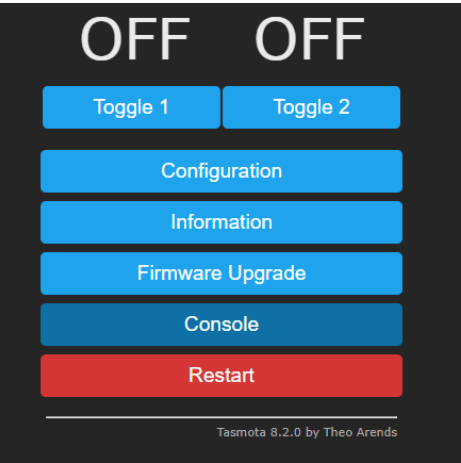
Unlocking Your Devices - Custom Firmware & Sensors

Gaining control, privacy, and power with Tasmota, ESPHome, and DIY hardware.

Before



After





Why Not Just Use Devices Out of the Box?

The Cloud Trap

Devices require manufacturer apps and internet. If servers fail, your device becomes useless.

Privacy Concerns

Where is your data going? Who is listening to your commands?

Speed Issues

Internet reliance adds latency. Your signal travels to distant servers and back.

Interoperability

Different apps can't communicate. You're locked into one ecosystem.

The DIY Spectrum

Three levels of customization, from beginner to expert.

1

Level 1: Flashing Existing Devices

Replace factory firmware on devices like Shelly or Sonoff.

Easiest way to gain local control.

2

Level 2: Using DIY Dev Boards

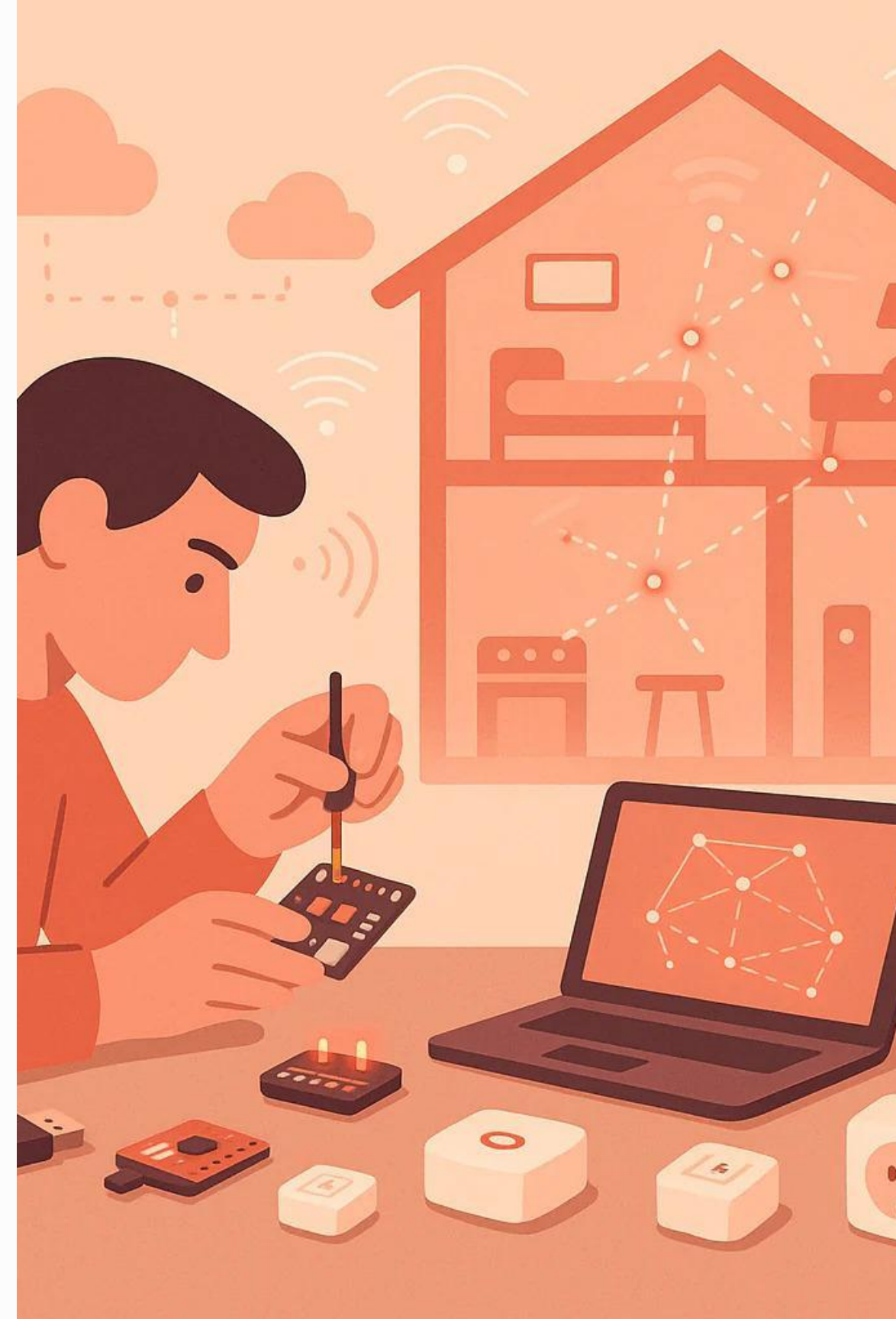
Build custom devices with ESP32, ESP8266, or Arduino.

Perfect for sensors that don't exist commercially.

3

Level 3: Full Custom Firmware

Write your own code from scratch in C++. For highly specific, power-optimized, or complex projects.



Your Customization Toolkit

Dive deeper into the powerful open-source solutions that empower you to take control. These firmware options align perfectly with the first two levels of the DIY Spectrum, enabling local control and endless possibilities for your smart devices.



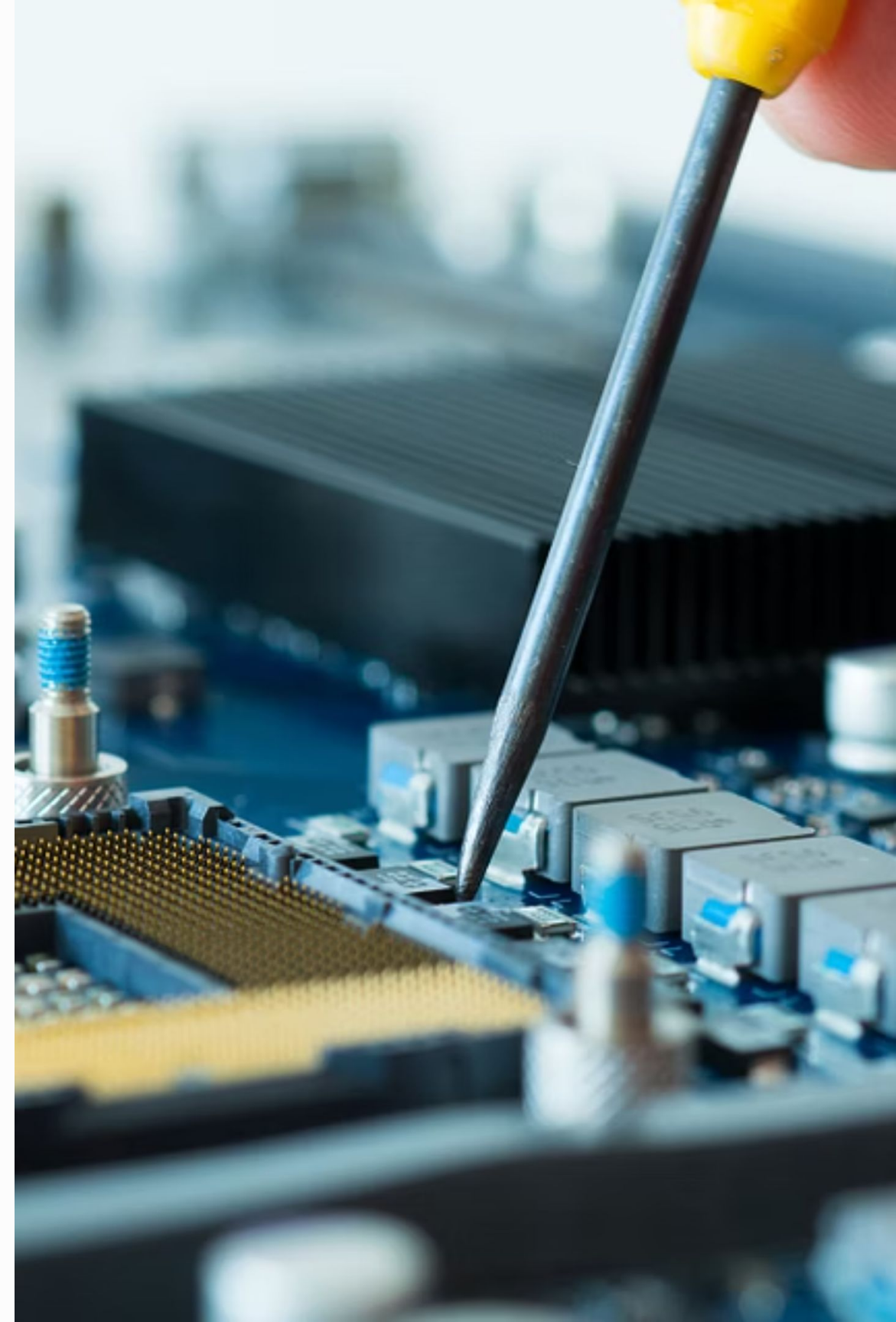
Tasmota: Flash & Go

Unlock off-the-shelf devices with ease.



ESPHome: Build Your Own

Craft custom sensors and devices.



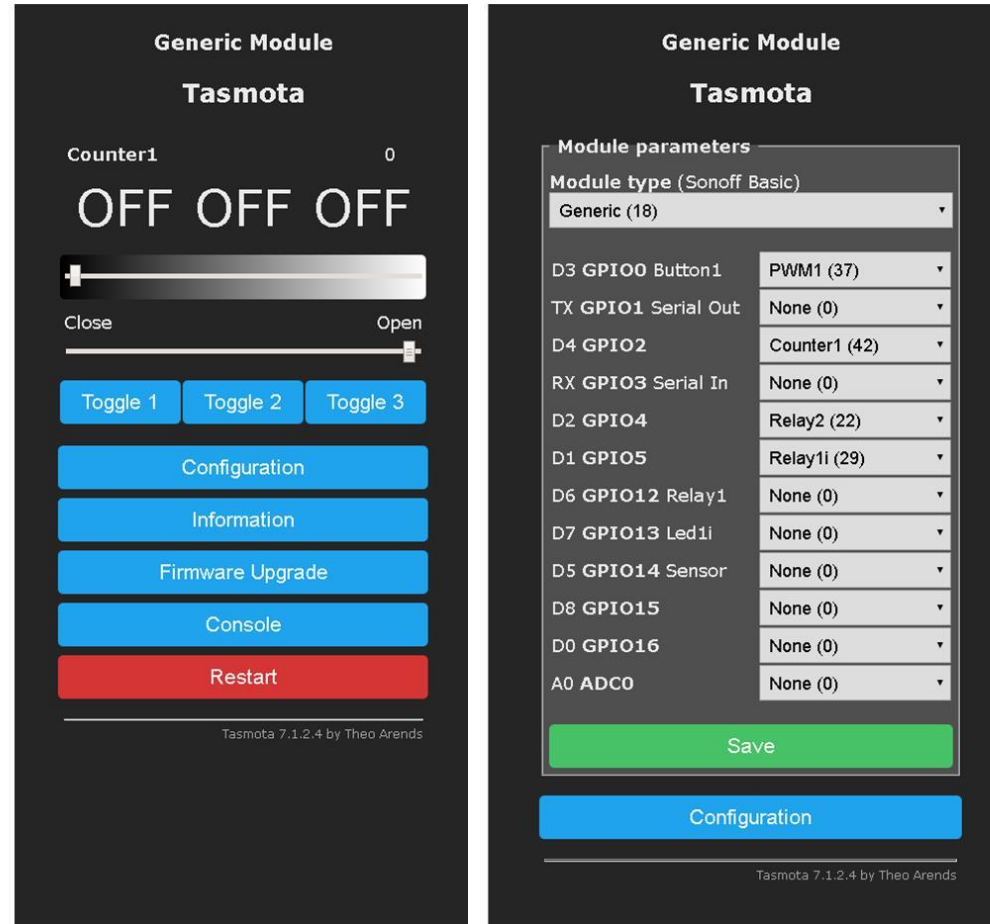
Meet Tasmota



V 4.1.1

Shutter¹

D1: Relay1i = EN
D2: Relay2 = DIR
D3: PWM1 = STP
D4: COUNTER1 = connected to D3/PWM1



A Versatile Firmware Solution

Tasmota offers a free, pre-compiled firmware solution compatible with hundreds of smart devices.

- Single flash and web-based configuration for ease of use
- Requires no coding, relying purely on intuitive configuration
- Rich feature set including timers, rules, and scripting capabilities
- Seamless integration via MQTT protocol

Universal

A single firmware solution compatible with a vast range of devices.

Easy Setup

Intuitive web-based configuration eliminates the need for coding.

Powerful

Delivers advanced features for comprehensive home automation.

Meet ESPHome



Custom Firmware Made Simple

Create firmware tailored to your specific needs using simple YAML files.

- Write YAML describing your device setup
- ESPHome compiles custom firmware automatically
- Deep integration with Home Assistant
- OTA updates over Wi-Fi

"I have an ESP32. A temperature sensor is on pin 4. A relay is on pin 5."

```
1 esphome:
2   name: demo-secur
3
4 esp32:
5   board: esp32dev
6   framework:
7     type: arduino
8
9 # Enable logging
10 logger:
11
12 # Enable Home Assistant API
13 api:
14   encryption:
15     key: "X9nD2DovL/bHekp7Cdomk7i1SD1CaCaXGR79BiwrpvU="
16
17 ota:
18   password: "2953a44c9a03d73ce7b92d6e91cb08ee"
19
20 wifi:
21   ssid: !secret wifi_ssid
22   password: !secret wifi_password
23
24 # Enable fallback hotspot (captive portal) in case wifi connection fails
25 ap:
26   ssid: "Demo-Seur Fallback Hotspot"
27   password: "0izS9qwbXZ7x"
28
29 captive_portal:
30
```

Tasmota vs. ESPHome

Which firmware should you choose?

Tasmota

What: Universal, feature-packed firmware

Interface: Web-based configuration

Integration: MQTT with Home Assistant

Best for: Quickly flashing existing devices like plugs and switches

ESPHome

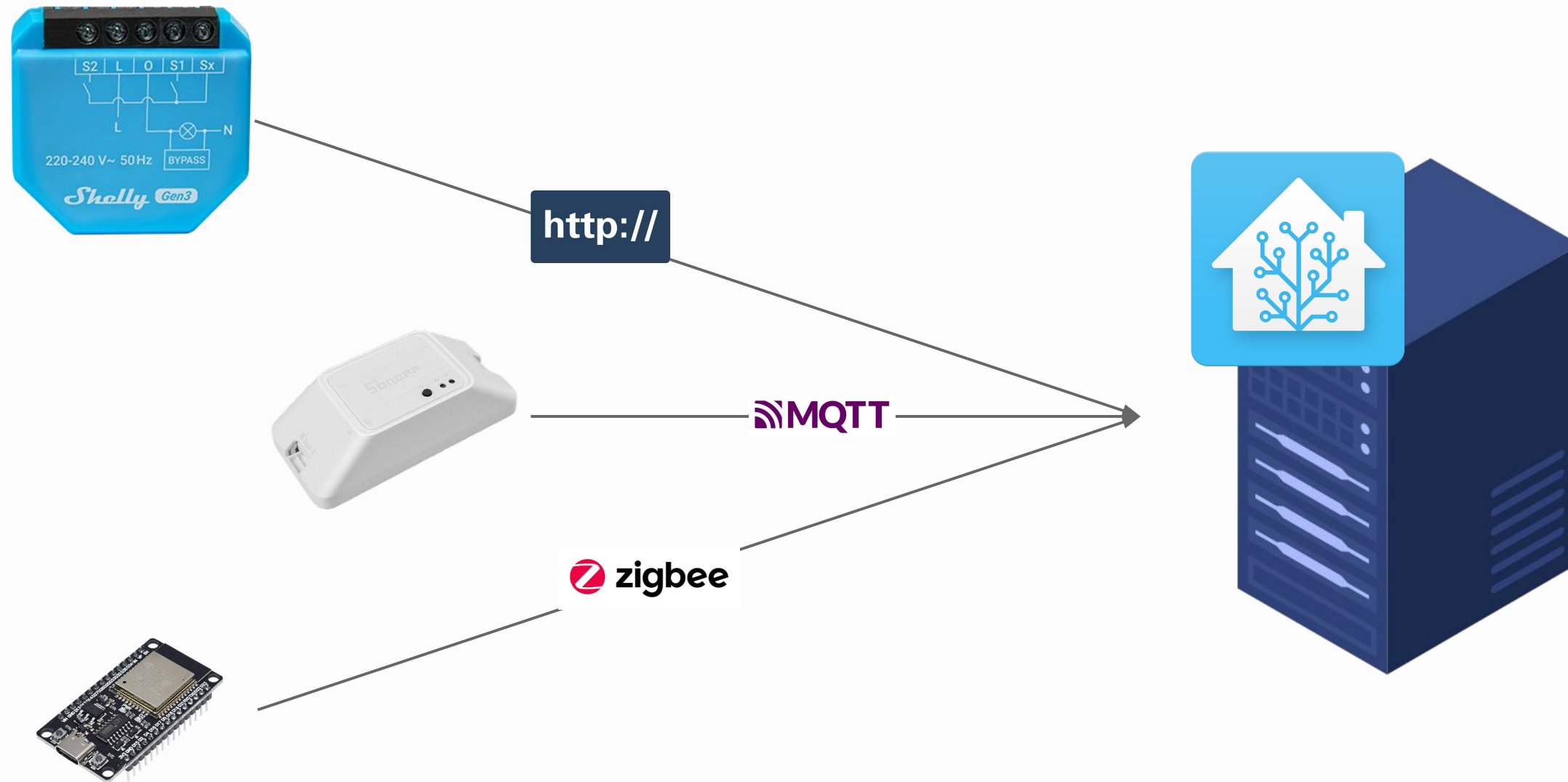
What: Custom firmware builder

Interface: YAML file configuration

Integration: Native, deep Home Assistant integration

Best for: Creating custom sensors with seamless automation

How Devices Talk: Communication Protocols



The "Languages" of IoT

Two Main Categories:

1

IP-Based (Wi-Fi/Ethernet)

- Devices that connect directly to your home network (e.g., ESPHome, Tasmota)

2

Mesh-Based (RF)

- Low-power devices that create their own network (e.g., Zigbee, Z-Wave)

Key Factors: Power Consumption, Range, Speed, and Reliability.

Protocol 1: HTTP

How it Works

Devices use standard web requests (GET/POST) to communicate directly with Home Assistant's API.

Example: A temperature sensor sends data like "21.5" to `/api/states/sensor.my_temp`.

Benefits (Pros)

Familiar & Simple: Easy to understand and implement due to HTTP's internet foundation.

Quick Setup: Ideal for basic "webhooks" triggering actions in Home Assistant.

Drawbacks (Cons)

High Overhead: Each message carries significant extra data, reducing efficiency.

Inefficient for Scale: Resource-intensive for Home Assistant when many devices send frequent updates.

Protocol 2: MQTT

How it Works

Devices publish messages to topics on a central broker. Home Assistant subscribes to these topics to receive updates.

Example: A temperature sensor publishes "21.5" to topic "home/living_room/temperature". Home Assistant subscribes to this topic.

Benefits (Pros)

Lightweight & Fast: Minimal overhead makes it ideal for IoT devices with limited resources.

Decoupled Communication: Devices don't need to know about each other, just the broker.

Reliable Delivery: Built-in quality of service levels ensure message delivery.

Standard for DIY: Default choice for Tasmota and most custom firmware.

Drawbacks (Cons)

Requires Broker: Need to run and maintain an MQTT broker (like Mosquitto).

Learning Curve: Topic structure and QoS levels require understanding.

Network Dependency: All communication flows through the central broker.

Protocol 3: Zigbee

How it Works

Devices form a self-healing mesh network using low-power radio signals. Each device can relay messages, extending range and reliability.

Example: A door sensor sends data through nearby devices to reach a USB coordinator connected to Home Assistant.

Benefits (Pros)

Ultra Low Power: Battery devices can last years without replacement.

Self-Healing Mesh: If one device fails, messages automatically route through others.

No Wi-Fi Congestion: Uses separate 2.4GHz channels, doesn't interfere with your network.

Standardized: Works with devices from different manufacturers.

Drawbacks (Cons)

Requires Coordinator: Need a USB Zigbee coordinator (like ConBee II or Sonoff Zigbee).

Setup Complexity: Pairing and network management can be more involved.

Range Limitations: Individual hops are shorter than Wi-Fi, needs mesh density.

Putting It All Together

Let's see how these protocols and firmwares work together in a real-world scenario:



Detect Motion

A **Zigbee sensor** detects motion in a room and sends a low-power signal.



Trigger Automation

Your **Home Assistant hub** receives the signal and runs an automation: "If motion AND dark..."



Publish Command

Home Assistant publishes a "turn_on" command to a specific **MQTT topic**.



Activate Device

Your **Tasmota smart plug**, subscribed to that topic, instantly turns on the light!

Thank You

Michalis Raptakis

Backend Engineer @Fairlo



Resources

- Tasmota: <https://tasmota.github.io/docs/>
- ESPHome: <https://esphome.io/>
- ESPHome vs Tasmota: <https://homeautomationtalks.com/esphome-vs-tasmota-which-one-is-right-for-you/>
- Tasmota Supported Devices Repository: <https://templates.blakadder.com/>
- ESPHome Devices: <https://devices.esphome.io/>
- HTTP: <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- MQTT: <https://mqtt.org/>
- Zigbee: <https://csa-iot.org/all-solutions/zigbee/>