AURORA · RESEARCH — CONTROL4 ZIGBEE LAB

Instalación y demostración end-to-end — Control4 SR260 → Sonoff vía Home Assistant

Investigación: replicación del fork `pvanbaren/zha-device-handlers@control4` sobre hardware real. **Fecha de la prueba end-to-end:** 2026-08-11. **Resultado:** ✅ un control remoto **Control4 SR260** maneja un módulo **Sonoff ZBMINI-L2** — dos marcas mutuamente incompatibles — con **Home Assistant + ZHA + los quirks** como único puente, **sin controlador Control4**.

Este documento es el registro paso a paso de la instalación y la demo, sin omitir nada: flasheo del CatSniffer, adopción de puertos, contenedor de HA, los quirks y la traducción del protocolo, la instalación eléctrica, el emparejamiento, la automatización (trigger + acción), los archivos que se montaron en cada reinicio, todos los obstáculos y sus soluciones.

Los documentos hermanos (`01` – `04`) cubren el análisis de código, la referencia de protocolo, los hallazgos confidence-tagged y el flasheo del CatSniffer en detalle. Este `05` es la narrativa de la puesta en marcha completa.

---

0. Resumen ejecutivo

Qué se probó	Que los quirks Control4 (v1, monkey-patch sobre zigpy/zha) funcionan sobre hardware físico y sobre un radio **EZSP** (no solo el ZNP donde se desarrollaron).
Cadena demostrada	Botón rojo Control4 (SR260) → protocolo propietario C4 sobre Zigbee → quirks traducen → `zha_event` en HA → automatización → `switch.toggle` → Sonoff conmuta.
Hardware Control4	SR260 (remoto) + EA-1 (controlador, presente pero no usado como puente). **No** se usó ningún módulo de iluminación Control4.
Coordinador Zigbee	Sonoff ZBDongle-E (EFR32MG21, EZSP/bellows).
Dispositivo controlado	Sonoff ZBMINI-L2 (relé Zigbee 3.0 estándar, marca incompatible con Control4).
Sniffer	Electronic Cats CatSniffer v3.1 (CC1352P7).
Stack HA	Home Assistant 2026.7.1 · zha 2.0.0 · zha-quirks 2.1.0 · zigpy 2.0.0 · Python 3.13.2.
Hallazgos nuevos	(1) ZHA 2.x exige los quirks **aplanados** al nivel superior; (2) el SR260 se provisiona sobre **EZSP** igual que sobre ZNP; (3) `default_config:` **no** incluye `automations.yaml` — sin el `include` la automatización nunca se carga.

---

1. Objetivo y alcance

El fork `control4` de zha-device-handlers afirma poder manejar hardware Zigbee de Control4 desde Home Assistant sin un controlador Control4. Hasta esta sesión eso estaba validado **solo en banco** (sin hardware):

los 5 monkey-patches instalaban, los 7 modelos registraban, 21 tests de parsing pasaban. Todo el bloque "Fase 4 — Replicación en HA" de `03-FINDINGS.md` estaba en **PENDING**.

El objetivo de esta sesión fue cerrar ese bloque con hardware real y llegar a una **demostración cross-brand**: que apretar el botón rojo Control4 encienda/apague un Sonoff. Es el caso más exigente y más vistoso, porque obliga a que toda la cadena —radio, intercepción, resolución de modelo, provisión, decodificación de botón, evento, automatización y actuación— funcione junta.

---

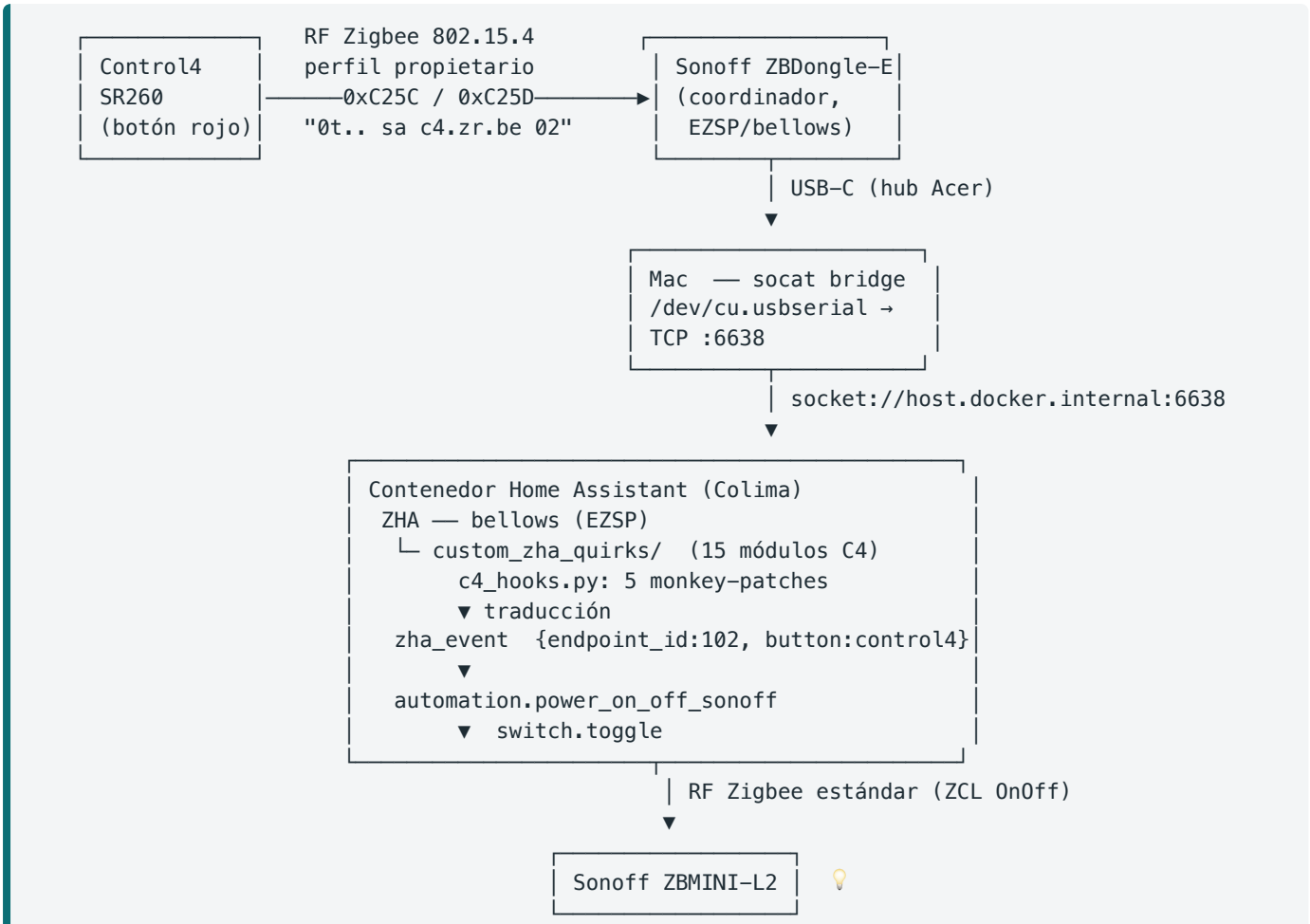
2. Inventario de hardware

El SR260 (izquierda), el coordinador Sonoff ZBDongle-E con antena (arriba a la derecha, sobre el hub Acer) y el controlador EA-1 (caja negra). El botón rojo con el "4" es el que dispara toda la demo.

Rol	Equipo	Detalle técnico
Remoto Control4	**SR260**	IEEE `00:0f:ff:00:00:60:a7:3c` (OUI Control4 `00:0f:ff`). End device *sleepy*.
Controlador Control4	**EA-1**	Presente como referencia; **no** se usó como puente en esta demo.
Coordinador Zigbee	**Sonoff ZBDongle-E**	EFR32MG21, +20 dBm, antena SMA. Radio **EZSP** (bellows). Firmware de fábrica **EmberZNet 6.10.3** (viejo).
Dispositivo controlado	**Sonoff ZBMINI-L2**	Relé Zigbee 3.0 "no-neutral". Marca incompatible con Control4 — es el experimento de control.
Sniffer	**CatSniffer v3.1**	CC1352P7 + RP2040 + SX1262. Firmware TI sniffer.
Host	**MacBook (Apple Silicon)**	macOS. Docker vía Colima (sin passthrough USB).

---

3. Arquitectura del montaje



Puntos clave de la topología:

- **El SR260 y el ZBMINI están en la misma red Zigbee**, formada por el ZBDongle-E. El EA-1 no participa en la ruta.
- **Docker en macOS no pasa USB** (el engine corre en una VM). Por eso el radio se expone por TCP con `socat` y ZHA lo consume como `socket://`.
- **Los quirks viven dentro del contenedor** en `custom_zha_quirks/`, cargados por ZHA al arranque, y parchean zigpy/zha en caliente.

---

4. Fase 1 — CatSniffer: flasheo del firmware

El CatSniffer v3.1 tiene **tres chips** y por eso el flasheo es en dos etapas. Ver `04-CATSNIFFER-FLASHING.md` para el detalle completo; acá el resumen operativo tal como se ejecutó.

El CatSniffer v3.1 (CC1352P7 + antena de 2.4 GHz) alimentado por USB. Aclaración: la placa con antena que se ve titilar en azul en el video es este CatSniffer — el LED azul parpadeante indica actividad de radio/captura.

```
USB-C ── RP2040 ──┬── UART+JTAG ── CC1352P7 ──┐
                  │                           ├── RF switch ── SMA (antena 2.4 GHz)
                  └── SPI ──────── SX1262 ────┘
```

4.1 Etapa 1 — passthrough en el RP2040 (una vez)

El RP2040 es el MCU host; no es el radio. Debe correr **SerialPassthroughwithboot**, que es el sketch que escucha la **cadena mágica** `<boot>` a 921600 baud y maneja por sí mismo los pines de bootloader del CC1352.

- La placa llegó ya con ese sketch (verificado indirectamente: el flasheo del CC1352 solo puede tener éxito si el RP2040 responde a `<boot>`), así que **no hizo falta reflashear** el RP2040.
- Si hubiera hecho falta: doble-click en `reset1` → monta el volumen `RPI-RP2` → copiar `SerialPassthroughwithboot.uf2` a la raíz.

4.2 Etapa 2 — firmware del CC1352P7 (por rol)

El CC1352 se flasheó como **sniffer TI** (`sniffer_fw_CC1352P_7_v1.10.hex`) con:

```
./tools/flash_cc1352.sh sniffer
```

que envuelve el `catnip_uploader` de Electronic Cats.

⚠️ **Regla dura (obstáculo #1): nunca** llamar `cc2538-bsl` directo. Entrar al bootloader del CC1352 en esta placa **no** es un flag de cc2538-bsl: depende de que el sketch passthrough del RP2040 escuche la cadena mágica `<boot>`. Hand-rollear ese handshake con flags inventados es cómo se brickea el radio. `catnip_uploader` hace la secuencia completa (abrir a 921600 → enter-bootloader → `cc2538 -e -w -v` → exit-bootloader).

⚠️ **Obstáculo #2 — el sniffer es una puerta de una sola vía por serial.** Flashear *hacia* `sniffer_fw` funciona; flashear *desde* él por serial no (el build TI deja cerrado el backdoor del ROM bootloader). La única vuelta a firmware coordinador es por **SWD**, que exige un doble-click físico en `reset1`. Por eso se decidió usar un **radio dedicado** (el ZBDongle-E) como coordinador, en vez de reflashear el CatSniffer ida y vuelta.

⚠️ **Obstáculo #3 — el mensaje de éxito del flasher miente.** `catnip_uploader.send_firmware()` hace `return True` siempre, incluso si el subproceso `cc2538` falla. `tools/flash_cc1352.sh` fue endurecido para grepear `ERROR:` / `Timeout waiting` y salir con código ≠ 0. **Verificar siempre con el PING** (`40 53 40 00 00 40 40 45` a 921600 → responde con `@S`), no con el mensaje.

El CatSniffer (sobre el mueble, el LED/antena azul que titila en el video es este dispositivo) capturando 802.15.4 mientras se opera el SR260. Rango dinámico medido en las capturas: −38 dBm a −90 dBm con FCS correcto — prueba de que la antena de 2.4 GHz y el RF switch están en el camino correcto.

4.3 Adopción de puertos — quién es dueño del USB

⚠️ **Obstáculo #4 — Parallels le roba el CatSniffer a macOS.** Al enchufar la placa con una VM corriendo, **no aparece** ningún `/dev/cu.*` aunque el dispositivo enumere bien: Parallels lo pasa exclusivamente a la VM Windows (`UsbExclusiveOwner = prl_vm_app` en `ioreg`). Solución: **desactivar el auto-connect** de Parallels para "Arduino / RaspberryPi Pico". Regla de puertos:

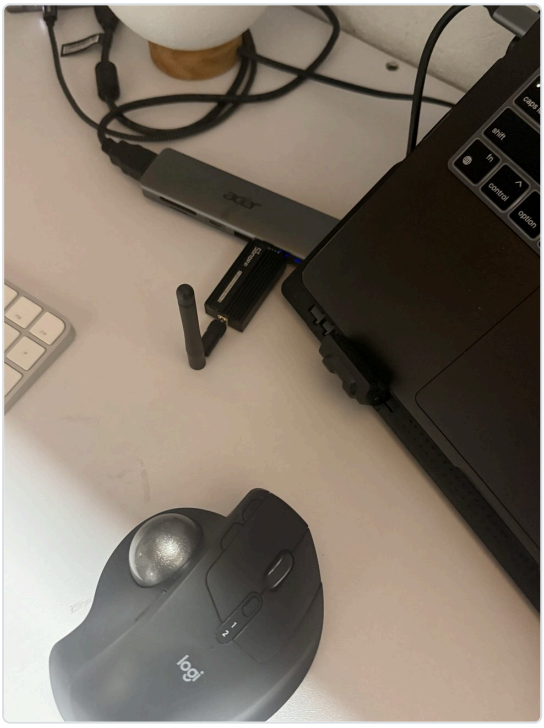
Fase	Dueño del USB	Motivo
Flasheo (CC1352)	**macOS**	`cc2538-bsl` necesita el nodo `/dev/cu.*`
Sniffing	**macOS**	`pycatsniffer_bv3` habla el protocolo TI directo en macOS
Coordinador HA	**macOS**	el bridge socat necesita el nodo serial

> ⚠️ **Obstáculo #5 — `system_profiler SPUSBDataType` no es confiable acá:** devuelve un árbol USB vacío en este contexto, indistinguible de "nada enchufado". Usar `ioreg -p IOUSB` (imprime IDs en decimal).

---

5. Fase 2 — Adopción de puertos y bridge serial del coordinador

El coordinador de producción es el **ZBDongle-E**, no el CatSniffer.



El Sonoff ZBDongle-E (con antena) enchufado al hub Acer USB-C, que va a la MacBook. Este es el radio que forma la red y habla con el SR260 y el Sonoff.

Como Docker/Colima en macOS no pasa USB, el radio se expone por TCP:

```
./tools/serial-bridge.sh /dev/cu.usbserial-1140
```

> ⚠️ **Obstáculo #6 — la sintaxis de baudios de socat.** Este build de socat rechaza `b115200`. El script `tools/serial-bridge.sh` se corrigió para usar `ispeed=115200,ospeed=115200,...`:
>
> `socat -d -d TCP-LISTEN:6638,reuseaddr,fork,nodelay \ FILE:/dev/cu.usbserial-1140,ispeed=115200,ospeed=115200,raw,echo=0,clocal=1,cs8,parenb=0,cstopb=0,crtscts=0`

En el flujo de configuración de ZHA se usa entonces:

- **Radio type:** EZSP (Silicon Labs EmberZNet) — porque el ZBDongle-E es EFR32MG21.
- **Serial path:** `socket://host.docker.internal:6638`
- **Baud:** 115200

> **Por qué EZSP y no ZNP:** el fork se desarrolló sobre ZNP, pero se verificó por inspección que los 5 patches viven en clases base que **tanto bellows (EZSP) como zigpy-znp heredan sin override**. Patch 4 (`packet_received`) está en `ControllerApplication` ; Patch 2 (`custom_profile_packet_received`) en `Device` . Conclusión validada hoy con hardware: **funcionan idénticos sobre EZSP.**

---

6. Fase 3 — Home Assistant + ZHA (contenedor)

6.1 Runtime fuera de iCloud

> ⚠️ **Obstáculo #7 — el estado de runtime no puede vivir en iCloud Drive.** HA mantiene bases SQLite vivas (`zigbee.db` , `home-assistant_v2.db`) e iCloud las sincroniza/desaloja, corrompiéndolas. `bootstrap.sh` pone el runtime en `~/c4lab` ; solo el código y los docs quedan en la carpeta sincronizada.
>
> *(Este mismo problema mordió al renderizar el video de esta demo: escribir el mp4 directo a la carpeta iCloud lo dejó truncado sin `moov atom` . Se renderiza a disco local y recién ahí se copia.)*

6.2 Contenedor y stack pinneado

> ⚠️ **Obstáculo #8 — Docker en macOS.** `brew install docker` es solo el cliente; no hay kernel Linux. El engine vino de **Colima** (Apache-2.0, sin admin). Hubo que limpiar restos de Docker Desktop en `~/.docker/config.json` (`credsStore: desktop` y `currentContext: desktop-linux` rompían todo).

`docker compose up -d` levanta `c4lab-ha` . Versiones dentro del contenedor, **idénticas al venv de banco** (por eso los tests offline prueban el mismo código):

paquete	versión
homeassistant	2026.7.1
zha	2.0.0
zha-quirks	2.1.0
zigpy	2.0.0

> ⚠️ **No flotar HA a `:stable` .** zigpy 2.1.0 rompe el fork (mueve la API de quirks v1 a `zhaquirks.legacy` , que el fork no tiene). Pinnear a 2026.7.x y re-correr `validate_quirks.py` + los tests ante cualquier upgrade.

6.3 `configuration.yaml` — cómo ZHA encuentra los quirks

```
default_config:

# (agregado en esta sesión — ver §9, obstáculo del include)
automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml

zha:
  custom_quirks_path: /config/custom_zha_quirks
```

Se verificó que `zha: custom_quirks_path:` **sigue siendo honrado** en HA 2026.7 (era una pregunta abierta en `01-CODE-ANALYSIS.md` §8).

---

7. Fase 4 — Los quirks y la traducción del protocolo

7.1 Qué son los quirks

15 módulos Python (7.733 líneas) que enseñan a ZHA a hablar el dialecto Control4. El corazón es `c4_hooks.py`, que instala **5 monkey-patches** al importarse:

#	Objetivo (parche sobre)	Para qué
1	`zigpy.endpoint.Endpoint.initialize`	Los endpoints C4 nunca responden `Simple_Desc_req`. Inyecta descriptores fijos (EP 2/196/197) para que el interview termine.
2	`zigpy.device.Device.custom_profile_packet_received`	Atrapa frames de perfil C4 **antes** de que zigpy los descarte; los rutea y refresca `last_seen`.
3	`zigpy.quirks.get_device`	Fuerza el quirk de modelo específico a ganar sobre el match por firma, aun con `manufacturer=None`.
3b	`DeviceRegistry.resolve()` de ZHA 2.x	El camino que ZHA 2.x realmente usa (Patch 3 solo sería código muerto en 2026.7).
4	`ControllerApplication.packet_received`	Intercepta frames C4 **broadcast** para sniffear el string de modelo de dispositivos aún no identificados.

Todo parche gatea sobre el prefijo IEEE `00:0f:ff` (OUI Control4): los quirks no pueden afectar a un dispositivo no-Control4. Buena propiedad de contención.

Registro confirmado en el arranque de hoy:

```
[c4_hooks] === C4 QUIRK FILE LOADED (multi-device) ===
[c4_hooks] C4: Installed endpoint interview patch
[c4_hooks] C4: Installed custom_profile_packet_received patch
[c4_hooks] C4: patched zigpy.quirks.get_device
[c4_hooks] C4: patched zhaquirks.ZHA_DEVICE_REGISTRY.resolve (class DeviceRegistry)
[c4_hooks] C4: Installed broadcast packet intercept patch
[zhaquirks] Loaded custom quirks.
```

7.2 La traducción — protocolo propietario → evento estándar

Control4 **no usa ZCL** para control. Tunela un lenguaje de comandos **ASCII** sobre Zigbee APS en tres perfiles propietarios:

Perfil	Constante	Uso
`0xC25C`	`C4_PROFILE_BUTTON`	Eventos de botón, LCD/listas
`0xC25D`	`C4_PROFILE_NETWORK`	Anuncios de red, broadcast de modelo
`0xC25E`	`C4_PROFILE_OUTLET`	Outlet dual (no aplica acá)

Formato de frame (un comando por frame, CRLF, latin-1):

```
<tipo><seq> <verbo> <namespace> [args...]
```

El botón rojo del SR260 (namespace `c4.zr.*`) emite en la práctica:

```
c4.zr.bb 02   → button begin (press)   botón 0x02 = "control4"
c4.zr.be 02   → button end   (release) botón 0x02 = "control4"
```

Los quirks mapean cada tecla física a un **endpoint virtual** `100 + button_id`. Para el botón rojo (`control4` , id `0x02`):

```
EP = 100 + 0x02 = 102
```

y emiten un `zha_event` de Home Assistant:

```
event_type: zha_event
data:
  device_ieee: "00:0f:ff:00:00:60:a7:3c"
  endpoint_id: 102
  cluster_id: 64578          # 0xFC42 = C4_BUTTON_CLUSTER_ID
  command: remote_button_short_release
  args: { button: control4, endpoint_id: 102 }
```

Eso es la traducción: de `c4.zr.be 02` (propietario, indescifrable para HA) a un `zha_event` estándar que cualquier automatización puede escuchar. (Tabla completa de los 50 botones en `02-PROTOCOL-REFERENCE.md` §SR260 button IDs. Nota-trampa: `digit_0` es `0x30` , **después** de `digit_9` y `star` .)

7.3 Hallazgo nuevo — ZHA 2.x exige los quirks aplanados

> ⭐ **Hallazgo (obstáculo #9, el que costó más).** Con los 15 `.py` dentro de una subcarpeta `control4/` (como en el repo), ZHA 2.x cargaba **0 quirks**: su loader usa `pkgutil.walk_packages` , que **solo escanea el nivel superior** de `custom_zha_quirks/` y no desciende a subcarpetas sin `__init__.py` . Diagnóstico: `walk_packages(path=['/config/custom_zha_quirks'])` devolvía `[]` .
>
> **Solución:** aplanar — copiar los 15 `.py` directo a `custom_zha_quirks/` (sin subcarpeta). `bootstrap.sh` ahora aplana por defecto. Tras aplanar, `walk_packages` ve los 15 módulos y `c4_hooks` corre.

7.4 Resolución de modelo y el seed

Los dispositivos C4 reportan un modelo inútil/ausente al interview, así que el fork usa un **cache persistente IEEE→modelo** en `/config/.storage/c4_quirk_data.json`, poblado por `_c4_sniff_model()` cuando el dispositivo hace broadcast de su identidad (ZCL Report Attributes, cmd `0x0A`, attr `0x0007`, formato `c4:control4_light:C4-SR260`).

> ⚠️ **Obstáculo #10 — el huevo y la gallina.** Al emparejar, el SR260 solo mandaba heartbeats (len=9), **no** el Report Attributes con el modelo. Sin modelo, el quirk no aplicaba; sin quirk aplicado, el dispositivo no se activaba para mandar el modelo. En ZHA aparecía como `unk_model` / `unk_manufacturer`.
>
> **Solución:** *seedear* el cache a mano y reiniciar: `json {"00:0f:ff:00:00:60:a7:3c": "C4-SR260"}` Tras el seed + restart: el quirk aplicó → el SR260 se activó → mandó su Report Attributes → `_c4_sniff_model` lo capturó → corrió el handshake de provisión ("identity sent", "MTORR sent") → los botones empezaron a disparar.

---

8. Fase 5 — Instalación eléctrica del Sonoff ZBMINI-L2

El ZBMINI-L2 es el dispositivo controlado. Se instaló en la caja de una llave de pared real.

Probador de tensión sin contacto Klein Tools, destornilladores y pinza de punta — el kit para intervenir la caja de pared con seguridad (verificar ausencia de tensión antes de tocar).

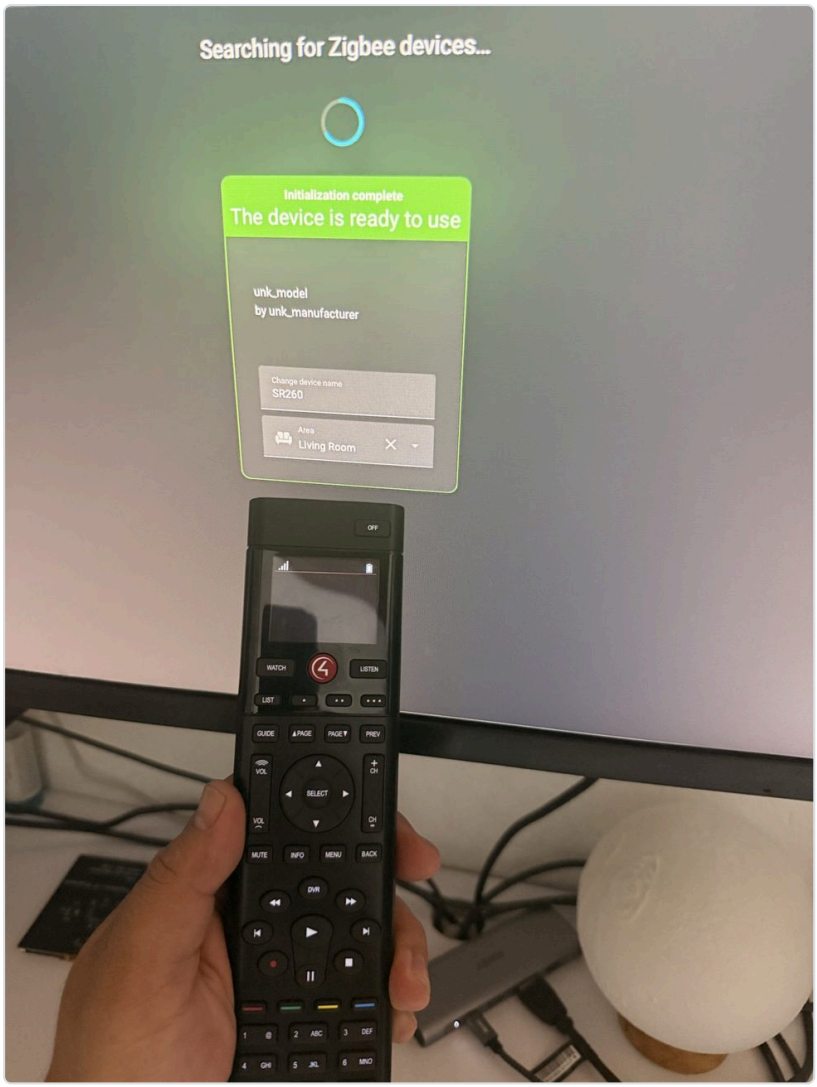
El Sonoff ZBMINI-L2 conectado en la caja: terminales L In / L Out (línea) y S1 / S2 (entradas de switch). El módulo va detrás de la tecla y conmuta la carga por Zigbee además del switch físico.

Este dispositivo es deliberadamente el **experimento de control**: es Zigbee 3.0 estándar, sin rareza Control4. Emparejarlo valida coordinador, formación de red, permit-join, creación de entidades en ZHA y el bridge socat/TCP **sin** los quirks en juego. Regla: **emparejar el ZBMINI antes que el SR260**.

---

9. Fase 6 — Emparejamiento en Home Assistant

Orden ejecutado: primero el coordinador, luego el ZBMINI (control), luego el SR260.



Emparejamiento del SR260: "Searching for Zigbee devices… / Initialization complete / The device is ready to use". Al principio aparece como `unk_model by unk_manufacturer` *— el problema del huevo y la gallina (§7.4) — hasta que el seed resuelve el modelo.*

La prueba: los tres dispositivos en la misma red Zigbee. SONOFF ZBDongle-E (coordinador, 44 entidades), SONOFF ZBMINIL2 (7 entidades) y SR260 — resuelto correctamente al modelo `C4-SR260` por el quirk — con 3 entidades.

Sobre el SR260 (sleepy device): tras cada reinicio de HA necesita despertar (apretar botones) para re-anunciar su modelo y re-provisionarse. El seed persistente hace que el quirk aplique al arranque; el handshake de provisión corre en cuanto el remoto despierta.

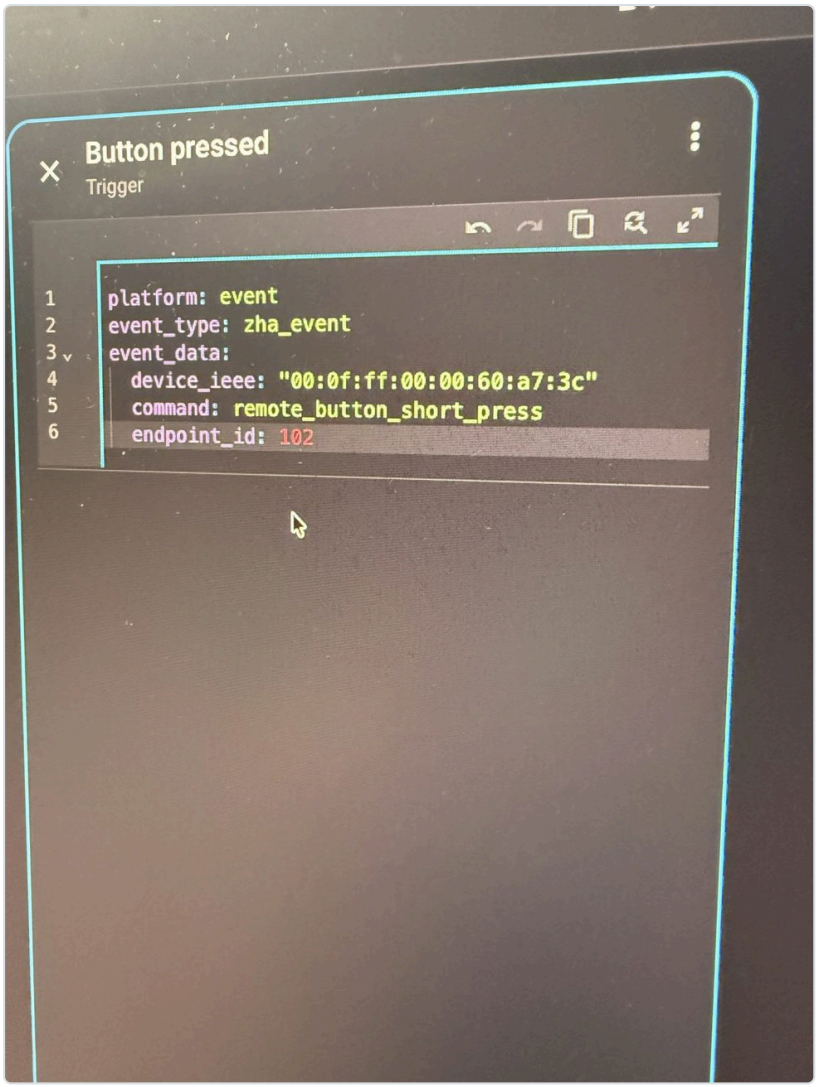
---

10. Fase 7 — La automatización (trigger + acción)

Aquí es donde se conecta el evento traducido con una acción real. Hubo varias iteraciones y un bug de fondo importante.

10.1 El trigger (disparador)

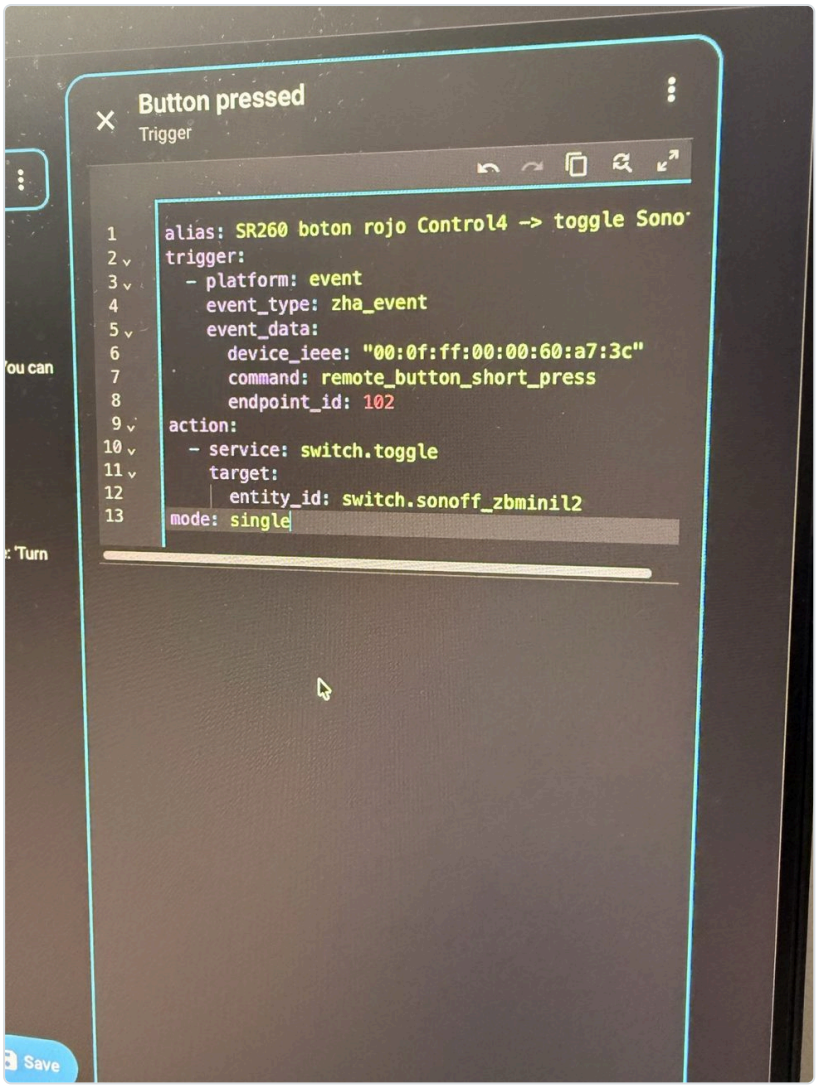
El trigger escucha el `zha_event` del botón rojo. En el editor de HA, el bloque de **trigger solo** (pestaña "Button pressed / Trigger") lleva estos 6 renglones:




```
platform: event
event_type: zha_event
event_data:
  device_ieee: "00:0f:ff:00:00:60:a7:3c"
  command: remote_button_short_press
  endpoint_id: 102
```

10.2 La acción (el "then do")

La acción conmuta el Sonoff:




```
action:
  - service: switch.toggle
    target:
      entity_id: switch.sonoff_zbminil2
mode: single
```

10.3 Qué YAML terminó funcionando

El archivo final que **sí** funcionó — guardado por la UI en `~/c4lab/ha-config/automations.yaml` — usa el formato de esquema nuevo (`triggers` / `actions` en plural) y dispara en el **release** del botón:

```
- id: '1786495992235'
  alias: Power ON/OFF SONOFF
  triggers:
    - trigger: event
      event_type: zha_event
      event_data:
        command: remote_button_short_release
        endpoint_id: 102
  conditions: []
  actions:
    - action: switch.toggle
      target:
        entity_id: switch.sonoff_zbminil2
  mode: single
```

> ⚠️ **Obstáculo #11 — errores de forma del YAML.** Pegar la automatización completa en el editor de **trigger-solo** da `expected str @ data['triggers'][0]['platform']` ; quitar de más deja `required key ['trigger']` . El trigger-solo lleva `trigger:` (no `platform:` en el esquema nuevo), `event_type` y `event_data` .
>
> ⭐⭐ **Obstáculo #12 — el bug de fondo (el que rompía todo en silencio):** `default_config:` levanta el *componente* de automatizaciones (por eso el editor funciona y muestra "Triggered"), **pero no incluye** `automations.yaml` . Sin la línea `automation: !include automations.yaml` en `configuration.yaml` , la automatización guardada **nunca se carga como entidad real**: el trigger se *previsualiza* en el editor pero **la acción jamás corre**, y la UI tira "New automation setup timed out".
>
> **Solución:** agregar los tres includes (`automation` / `script` / `scene`), crear los `scripts.yaml` / `scenes.yaml` vacíos, validar y reiniciar. Confirmación en el log: `[automation.power_on_off_sonoff] Initialized trigger Power ON/OFF SONOFF` y la entidad `automation.power_on_off_sonoff` pasó a estado `on` .

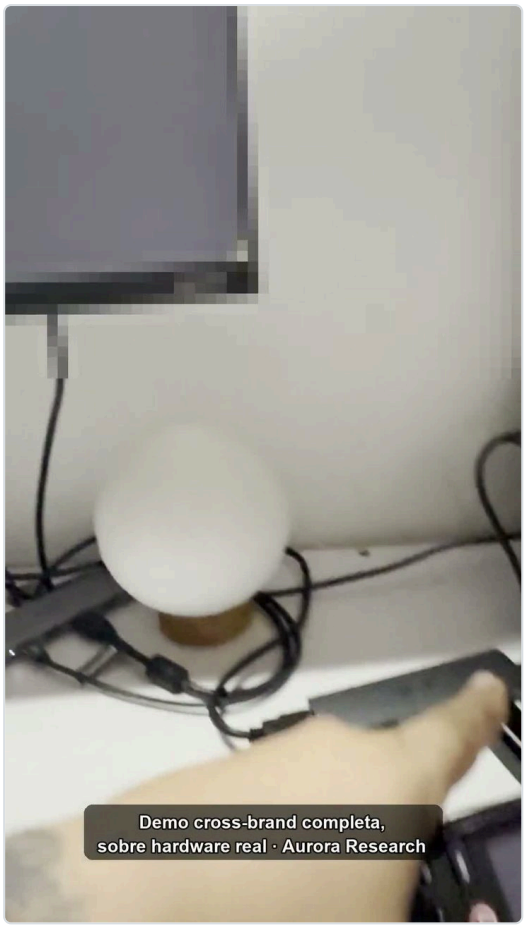
---

11. Fase 8 — Demostración end-to-end

Con la automatización cargada como entidad real, se apretó el botón rojo.

El video (sin audio, con subtítulos) recorre la cadena completa: el remoto, el módulo Sonoff instalado en la pared, y la reacción en Home Assistant.

Momento de la demo: se presiona el SR260 al lado de la Mac y el ZBDongle-E.



📼 ***video-demo-cross-brand.mp4*** *— demo cross-brand subtitulada (60 s, 1080×1920, sin audio; pantallas pixeladas por privacidad). El archivo de video acompaña este documento en* `docs/photos/` *.*

11.1 Lo que mostró el log

El trigger disparó y la acción corrió (traza real de hoy):

```
[automation.power_on_off_sonoff] Running automation actions
[automation.power_on_off_sonoff] Executing step call service      ← la acción SÍ corrió
```

El evento que la disparó, confirmado en "Triggering event detail":

```
endpoint_id: 102
cluster_id: 64578            # 0xFC42
command: remote_button_short_release
args: { button: control4, endpoint_id: 102 }
```

Coincide **exactamente** con `SR260_BUTTON_MAP`: `control4` → id `0x02` → EP 102.

11.2 El obstáculo final — RF intermitente

> ⚠️ **Obstáculo #13 — `device did not respond`.** De 4 pulsaciones, ~2 corridas de la acción fallaron con `Failed to send request: device did not respond` (el comando sale del coordinador pero el Sonoff no ACKea), y varias más quedaron rechazadas como `Already running` (con `mode: single`, cada pulsación se apila mientras la anterior espera el timeout de RF ~15 s). El toggle **desde el panel de HA funcionaba**, confirmando que el software estaba bien y el problema era del enlace RF.
>
> **Causa probable:** firmware viejo del ZBDongle-E (**EmberZNet 6.10.3**; los builds actuales son 7.4.x y esa serie tiene historial de TX flojo) y/o distancia. **Mitigación inmediata:** insistir/acercar los equipos. **Resultado:** una de las pulsaciones pegó y **el Sonoff conmutó — "prendió".** ✅

La demo cross-brand quedó cerrada: botón Control4 → Sonoff, dos marcas incompatibles hablando gracias a los quirks, sobre hardware real.

---

12. Archivos montados y reinicios de HA

Estructura del runtime (fuera de iCloud, en `~/c4lab/ha-config/`) y qué toca cada reinicio:

```
~/c4lab/ha-config/
├── configuration.yaml          default_config + includes + zha.custom_quirks_path + logger
├── automations.yaml            la automatización (cargada vía !include)
├── scripts.yaml   scenes.yaml  stubs vacíos ([]) que los includes exigen
├── custom_zha_quirks/          15 .py APLANADOS (no en subcarpeta) — §7.3
│   ├── c4_hooks.py             instala los 5 monkey-patches al importarse
│   ├── c4_helpers.py  c4_button_cluster.py  c4_display_cluster.py …
│   └── control4_remote.py  control4_switch.py  control4_dimmer.py …
└── .storage/
    ├── c4_quirk_data.json      SEED IEEE→modelo: {"00:0f:ff:…:a7:3c":"C4-SR260"}
    ├── core.restore_state      estados (se escribe al apagar; puede quedar viejo)
    └── zigbee.db               red Zigbee (SQLite viva — por eso NO en iCloud)
```

Qué pasa en cada **reinicio de HA**:

1. ZHA re-escanea `custom_zha_quirks/` (nivel superior) → importa los 15 módulos → `c4_hooks` reinstala los 5 patches.
2. El seed `c4_quirk_data.json` se lee al importar → el quirk del SR260 puede aplicar al arranque sin esperar un nuevo Report Attributes.
3. El SR260, al ser *sleepy*, se **re-provisiona** cuando despierta (primeras pulsaciones): re-manda modelo → handshake `identity sent` / `MTORR sent`.
4. `automation: !include automations.yaml` carga la automatización como entidad (`Initialized trigger Power ON/OFF SONOFF`).

> Nota de logging: se bajó el nivel a `warning` con overrides puntuales (`zhaquirks`, `c4_hooks`, `c4_button_cluster` en `info`; `homeassistant.components.automation` en `info`) para ver la cadena sin inundar el log — la inundación previa causaba el falso "setup timed out".

---

13. Registro consolidado de obstáculos

#	Obstáculo	Solución
1	Brickear el CC1352 con `cc2538-bsl` directo	Usar `catnip_uploader` (maneja la cadena mágica `<boot>`)
2	Sniffer = puerta de una sola vía por serial	Radio coordinador dedicado (ZBDongle-E); recuperación solo por SWD
3	El flasher reporta éxito aunque falle	Script endurecido: grep de `ERROR:` / `Timeout`, verificar con PING
4	Parallels roba el USB del CatSniffer	Desactivar auto-connect; regla de dueño por fase
5	`system_profiler` devuelve árbol USB vacío	Usar `ioreg -p IOUSB` (decimal)
6	socat rechaza `b115200`	`ispeed=115200,ospeed=115200,...`
7	Runtime en iCloud corrompe SQLite (y truncó el video)	Runtime en `~/c4lab`; render a disco local y luego copiar
8	Docker en macOS sin engine / restos de Docker Desktop	Colima; limpiar `~/.docker/config.json`
9	ZHA 2.x carga 0 quirks desde subcarpeta	**Aplanar** los .py al nivel superior
10	SR260 `unk_model` (huevo y gallina)	**Seed** `c4_quirk_data.json` + reiniciar
11	Errores de forma del YAML del trigger	Trigger-solo: `trigger:` / `event_type` / `event_data`
12	`default_config:` no incluye `automations.yaml`	Agregar `automation/script/scene: !include …`
13	RF intermitente (`device did not respond`)	Insistir/acercar; pendiente: actualizar firmware del ZBDongle-E

14. Estado final y próximos pasos

Cerrado hoy (2026-08-11): toda la "Fase 4 — Replicación en HA" que estaba en PENDING. El SR260 se une a la red (IEEE `00:0f:ff`), el quirk aplica sobre EZSP, la intercepción dispara, el modelo resuelve a `C4-SR260`, la provisión corre, los botones decodifican en sus endpoints virtuales, el `zha_event` sale correcto, la automatización corre y el Sonoff conmuta. **Demo cross-brand validada.**

Pendientes:

1. **Actualizar el firmware del ZBDongle-E** (EmberZNet 6.10.3 → 7.4.x) para eliminar la intermitencia de RF.
2. **Fase 3 — EA-1 ground truth:** sniffear al controlador oficial manejando el SR260 (requiere la network key del EA-1; ver `00-LAB-PLAN.md` §Phase 3).
3. Cerrar el resto del checklist de claims del SR260 (50 entidades de evento, sensor de batería, `long_press` repetido a ~100 ms, escritura LCD por cluster `0xFC47`, menú `show_list` / `close_list`, trigger `motion_wake`).

Anexo A — Galería de la instalación

Foto	Qué muestra
IMG_3813	CatSniffer v3.1 alimentado, LED azul
IMG_3814	ZBDongle-E en el hub USB-C → Mac
IMG_3815	ZBMINI-L2 cableado en la caja de pared
IMG_3816	Herramientas de instalación eléctrica
IMG_3817	SR260 emparejándose (unk_model)
IMG_3818	SR260 + ZBDongle-E + EA-1 en el escritorio
IMG_3819	Operando el SR260
IMG_3820	CatSniffer sniffeando (LED azul) durante el uso
IMG_3821	Presionando el botón rojo junto al coordinador
IMG_3822	**Los 3 dispositivos en ZHA (C4-SR260 resuelto)**
IMG_3823	Automatización completa (trigger + acción)
IMG_3824	Trigger YAML (event, endpoint 102)
video	**Demo cross-brand subtitulada (60 s)**

Anexo B — Datos de referencia rápidos

- **SR260 IEEE:** `00:0f:ff:00:00:60:a7:3c` · **OUI Control4:** `00:0f:ff`
- **Botón rojo:** `control4`, id `0x02` → endpoint virtual **102**
- **Cluster de botones:** `0xFC42` (64578 dec) · **LCD/listas:** `0xFC47`
- **Perfiles C4:** `0xC25C` botón · `0xC25D` red · `0xC25E` outlet · cluster wire `0x0001`
- **zha_event:** `command: remote_button_short_release`, `endpoint_id: 102`
- **Seed:** `/config/.storage/c4_quirk_data.json` → `{"00:0f:ff:00:00:60:a7:3c":"C4-SR260"}`
- **Bridge:** `socket://host.docker.internal:6638` (socat, 115200)
- **Stack:** HA 2026.7.1 · zha 2.0.0 · zha-quirks 2.1.0 · zigpy 2.0.0

Este trabajo es ingeniería inversa independiente sobre hardware propio. No es software oficial de Control4 / Snap One ni está avalado por ellos. Los detalles de protocolo provienen de tráfico observado y pueden estar incompletos.