Control Panel Test Procedure

Control Panel V2.0

Rev 0.0

Use with Control Panel Test Method Overview

Caleb Taylor | 2026-07-13

0. GENERAL SETUP

Scaffold-only testing: the Scaffold board test is §9, at the end of this document. If you are only testing a Scaffold board, complete §0 and go directly to §9 — it uses a known-good workcell board as the host.

0.1 Equipment

#	Item	Notes
1	Bench PSU, 0—30 V ≥ 10 A	PS4 / PS2 source
2	Bench PSU, 0—60 V ≥ 12 A	PS1 source (§2.3 high-voltage point)
3	Second low-voltage PSU, isolated output	−PS2 offset test (§2.2)
4	DMM, 6½-digit preferred	4-wire capable
5	Electronic load, ≥ 10 A	constant-current + dynamic modes
6	Oscilloscope, ≥ 50 MHz, 2 ch	Red Pitaya acceptable; AC-coupled 1× and 10× probes
7	Thermal camera or IR thermometer	shunt / switch / blade temperatures
8	USB3 SSD (known ≥ 300 MB/s), USB-C and USB-A cables	§5
9	USB2-only flash drive	§5.2, §6.3, §9.2
10	Gigabit Ethernet link-partner PC + 90 m Cat5e/6 cable	§1.5; also the iperf3 / SSH host
11	CAN FD partner node (any known-good MCP2518FD/FD-capable node) + twisted pair	§7
12	M.2 Key-E Wi-Fi module (AX210) + antennas	§6
13	USB-C sink tester or 5.1 kΩ-strapped breakout	§5.1
14	Imaged microSD card (current OS build), spare card	§1.2
15	USB-UART adapter, 3.3 V logic (debug console)	§0.3
16	Scaffold board (V2.x)	docked throughout — §0.2

0.2 Board configuration before testing

1. Verify jumper states: JP1—JP4 (spare GPIO to J2) as required by the test plan; JP5 (EEPROM WP) open; JP6 (CAN termination) noted — §7.3 tests both states.
2. Verify the ground-scheme jumper (GND↔EGND machine-pin jumper) state and record it on the Overview sheet. GND jumpers should be in place at the start of test. Sections that require a different state call it out.
3. F1 (PS4 fuse) fitted.
4. Image the microSD card with the current OS build. Preferred: Raspberry Pi Imager — Choose OS → Use custom, select the release `.img`, write, and let it verify. From a Linux PC instead: `sudo dd if=<release>.img of=/dev/sdX bs=4M conv=fsync status=progress` (find `sdX` with `lsblk` first — `dd` overwrites the target device without asking). Insert the imaged card in J5.
5. The Scaffold board remains docked on J2 for the entire procedure, with its GND jumpers placed, unless a step instructs otherwise.

0.3 Conventions

- **The CM5 boots automatically when PS4 power is applied** (an imaged SD card is fitted per §0.2). There is no separate "turn on" action; SW1 is only used for shutdown and wake-from-halt (§1.3).
- **Console access:** connect the USB-UART adapter to the debug UART header (115200 baud, 8N1, no flow control) and open a terminal program (e.g. Tera Term / PuTTY). Log in with the username and password printed in the OS image release notes. After §1.5 passes, SSH may be used instead: `ssh <user>@<board-ip>`.
- All shell commands are written exactly as typed. Commands shown with `sudo` require it; commands without `sudo` do not. If a command reports `command not found`, install it first with the line given in that step.
- GPIO operations use `pinctrl` (pre-installed; not legacy GPIO tools): `sudo pinctrl set <n> op dh` (drive high), `sudo pinctrl set <n> op dl` (drive low), `pinctrl get <n>` (read). CAM_GPIO0/1 are GPIO34/35 and are only accessible via `pinctrl`.
- I2C bus map: power meters = **i2c-1**; sensor slots = **i2c-2**; LED expander = **i2c-10**.
- Test scripts and firmware referenced as `/opt/cpw-test/…` ship with the OS image as the **cpw-test** package (naming to be finalized; sources in the `cpw-test/` folder of the project).
- Every step that measures something names the quantity and unit to record on the Overview sheet. Any FAIL stops the section and gets a note.
- Destructive characterization tests are collected in **Appendix A** and are run only on a designated engineering (sacrificial) board, never on a deliverable unit.

---

1. INITIAL DEVICE SETUP & FIRST BOOT

Sequence: apply first power (the CM5 boots by itself), verify the operator-facing basics, and bring the network up so the remaining sections can run over SSH.

1.1 First power — PS4 input

1.1.1 Set PSU #1 to 5.00 V, current limit 8 A, output OFF. Connect it to the J1 PS4 input (+PS4 / −PS4 poles, both WAGO contacts per pole).

1.1.2 Connect the scope before powering: CH1, AC-coupled, 10× probe, 20 mV/div, 10 µs/div; probe tip on the PS4 bulk capacitor positive pad, ground spring (not a long ground clip) on the capacitor negative pad.

1.1.3 Turn PSU #1 output ON. The CM5 begins booting immediately — wait for the login prompt on the debug console (§0.3), then leave the system idle at the prompt.

1.1.4 **Measure: idle ripple.** Read the peak-to-peak ripple voltage on CH1 with the system idle. Record the value in mV p-p.

1.1.5 **Apply a stress load** (full explanation of this step: `stress-step-explainer.txt`). Install the tool if needed: `sudo apt install -y stress-ng`. Then run: `stress-ng --cpu 4 --timeout 300s` — this loads all four CPU cores for 5 minutes to pull worst-case current on PS4.

1.1.6 **Measure: loaded ripple, two locations.** While stress-ng runs: (a) read mV p-p at the PS4 bulk capacitor (same point as 1.1.4) and record; (b) move the probe to the 5 V / GND test points beside the CM5 connectors (J4 area) and record mV p-p there as well.

1.1.7 **Measure: minimum supply voltage under load (droop).** Switch the scope to peak-detect (min/max) acquisition, DC-coupled, 1 V/div, and capture 60 s at the CM5 5 V test point while stress-ng runs. Record the **minimum instantaneous voltage** seen — this is the worst-case supply dip the CM5 experiences.

PASS: loaded ripple ≤ 100 mV p-p at both points; minimum ≥ 4.75 V at the CM5 test point.

1.2 SD & first boot

1.2.1 **Boot success.** Turn PSU #1 output OFF, wait 10 s, then ON. Watch the debug console (§0.3: USB-UART adapter on the debug header, 115200 8N1): boot messages scroll, ending at the login prompt. **Record: time from power-on to login prompt, in seconds.** Log in with the image's credentials.

1.2.2 **Boot loop.** Run `sudo reboot` (exact syntax — `reboot` alone is refused for a non-root user). The console session drops; wait for the login prompt to return and log in again. Repeat 10 times. **Record: number of successful boots out of 10.**

1.2.3 **Write soak.** Write a 5 GB test file to the SD card (the home directory is on the SD; do not use `/tmp`, which is RAM-backed):

```
dd if=/dev/zero of=$HOME/sdtest.bin bs=1M count=5120 conv=fsync status=progress
```

When it finishes, **record the write rate printed on the last line, in MB/s.**

1.2.4 **Read soak.** Clear the RAM cache so the read actually comes from the card, then read the file back:

```
sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
dd if=$HOME/sdtest.bin of=/dev/null bs=1M status=progress
```

Record the read rate printed on the last line, in MB/s. Then delete the test file: `rm $HOME/sdtest.bin`.

1.2.5 **Error check.** Run: `sudo dmesg | grep -iE "mmc|crc|timeout"` — this searches the kernel log for SD-interface errors. A healthy result shows only the normal mmc initialization lines from boot; any line containing `error`, `CRC`, or `timeout` that appeared during 1.2.3—1.2.4 is a FAIL. **Record: error count (expect 0).**

1.2.6 **SD power-gate observation.** The SD socket's power is switched by the RT9742 (U12). Because the running system's root filesystem is the SD card, the gate cannot be exercised from software while booted — observe it during power-up instead: scope the RT9742 output (SD VDD pin at the J5 socket; DC-coupled, 1 V/div, 5 ms/div, single-shot trigger rising 0.5 V) and power-cycle the board. **Record: the switched rail's rise time in ms and confirm a clean, monotonic rise with no oscillation.** Repeat across 3 power cycles.

PASS: 10/10 boots; zero mmc errors; gate rise clean on all 3 cycles.

1.3 PWR_BTN

1.3.1 Scope the SW1 button GPIO net (DC, 1 V/div, 5 ms/div, single-shot on falling edge). Press SW1 once. **Record: number of edges seen (expect one clean falling edge) and any contact-bounce duration in ms.**

1.3.2 With the OS running: short-press SW1. The console shows an orderly shutdown sequence ending in halt. **Record: shutdown clean yes/no.**

1.3.3 From halt (PSU still on): press SW1. The module wakes and boots to the login prompt. **Record: wake successful yes/no.**

1.3.4 Repeat 1.3.2—1.3.3 five times. **Record: successful cycles out of 5.**

PASS: 5/5 clean shutdown + wake cycles.

1.4 nPWR_LED

1.4.1 Observe the buffered power LED (driven by Q1) in each state and complete the truth table on the Overview sheet: (a) PSU on / CM5 halted, (b) CM5 booting, (c) CM5 running, (d) CM5 shut down via SW1. **Record: LED on/off for each of the four states.**

PASS: table complete and identical across 3 power cycles.

1.5 Ethernet link & SSH

Link-partner setup: a PC or server on the bench running an iperf3 server. Install there: `sudo apt install -y iperf3`, then run `iperf3 -s` and note its IP address (shown by `ip addr`). Install iperf3 on the CM5 the same way. All of §1.5 uses the 90 m Cat5e/6 cable.

1.5.1 **Lights.** Connect the 90 m cable to the link partner. Force each speed in turn and check the bezel LEDs against the design table (green/yellow per speed; activity LED flickers during `ping <partner-ip>`):

```
sudo ethtool -s eth0 speed 10 duplex full autoneg off
sudo ethtool -s eth0 speed 100 duplex full autoneg off
sudo ethtool -s eth0 speed 1000 duplex full autoneg off
```

Record: LED state correct at each speed, yes/no ×3.

1.5.2 **Gigabit link.** Restore auto-negotiation: `sudo ethtool -s eth0 autoneg on`. Then run `ethtool eth0` and confirm the `Speed:` line reads `1000Mb/s`. **Record: negotiated speed.**

1.5.3 **Throughput.** Run `iperf3 -c <partner-ip> -t 600` (10 minutes, CM5 transmitting), then `iperf3 -c <partner-ip> -t 600 -R` (receiving). **Record: both average throughputs in Mbit/s.**

1.5.4 **Error counters.** Run `sudo ethtool -S eth0 | grep -iE "err|fcs|drop"` . **Record: every non-zero counter (expect all zero).**

1.5.5 **Link robustness.** Run the flap loop exactly as written:

```
for i in $(seq 50); do
  sudo ip link set eth0 down; sleep 2
  sudo ip link set eth0 up; sleep 8
  ethtool eth0 | grep -q "1000Mb/s" || echo "FAIL $i"
done
```

Record: number of FAIL lines printed (expect 0 out of 50).

1.5.6 **SSH.** From the link-partner PC — the computer at the other end of the 90 m Ethernet cable, the same physical link under test — open an SSH session to the board: `ssh <user>@<board-ip>` (the board's IP is shown by `ip addr` on the debug console). Open the session before 1.5.3 and keep it open through 1.5.5. **Record: any stall or disconnect (expect none).** Background on what this step assumes: `ssh-step-explainer.txt` .

PASS: 1000 Mb/s on the 90 m cable; ≥ 900 Mbit/s each direction; zero error-counter movement; 50/50 flaps trained; SSH stable.

---

2. POWER

2.1 3V3 main ripple (U5)

2.1.1 **Measure: startup ramp.** Scope the +3v3 output capacitor bank at U5: single-shot, DC-coupled, 1 V/div, 2 ms/div, trigger rising at 1.0 V. Power-cycle the board and capture the 3v3 ramp.

2.1.2 Evaluate the capture: rise is monotonic (no step or plateau), overshoot ≤ 5 % (≤ 3.47 V peak), settles at 3.30 V ± 2 %. **Record: overshoot peak in V and final settled voltage in V.**

2.1.3 **Measure: steady-state ripple.** Reconfigure the probe for ripple: CH1, AC-coupled, 10× probe, 20 mV/div, ground spring on the U5 output capacitor bank. **Record ripple in mV p-p twice: system idle, and during `stress-ng --cpu 4 --timeout 120s` .**

PASS: startup clean per 2.1.2; steady ripple ≤ 50 mV p-p.

2.2 +PS2 relay (on–off)

Setup: PSU #1 → PS4 (5 V) as in §1.1. PSU #2 → PS2 input at J1, set 5.00 V, current limit 10 A. Electronic load connected across +PS2 / −PS2 at the docked Scaffold's machine contacts, set to 0 A.

2.2.1 **Command path.** On the CM5 run `sudo pinctrl set 35 op dh` — this drives CAM_GPIO1 high, commanding PS2 ON. Measure blade P2 with the DMM. **Record: voltage (expect 5 V).**

2.2.2 Run `sudo pinctrl set 35 op dl` — PS2 OFF. **Record: blade P2 voltage (expect 0 V).**

2.2.3 **Measure: switching timing.** Scope CH1 on the U6 isolator input (command side), CH2 on blade P2; single-shot on CH1 rising. Command ON as in 2.2.1 and capture. **Record: delay from command to output start in ms, and output 10—90 % rise time in ms.**

2.2.4 The output should be a controlled ramp in the low-ms range (TPS22990 slew control), not a step. Capture and **record the turn-OFF fall time in ms** the same way.

2.2.5 **Measure: on-resistance.** With the load at 1 A, then 5 A, then 10 A: DMM 4-wire from the U7 input pad to blade P2. **Record: voltage drop in mV at each current, and the computed resistance in mΩ.**

2.2.6 **Power-meter cross-check.** With PS2 ON and the load at 5 A: run `sudo i2cdetect -y 1` and confirm 0x44 ACKs, then read the PS2 channel: `sudo /opt/cpw-test/power-meter.py --rail PS2` . **Record: reported current in A (expect ≈ 5 A).** Set the load to 0 A.

2.2.7 **Isolation-domain switching.** Configure the Scaffold GND jumpers so −PS2 floats (lift the −PS2 bond; record the configuration on the sheet).

2.2.8 Connect PSU #3 between −PS2 and GND and set +2.50 V. Command PS2 ON: `sudo pinctrl set 35 op dh`. Measure directly across the blade pair with the DMM — red lead on +PS2_RLY (P2), black lead on −PS2, **not** on bench ground. **Record: blade voltage in V (expect ≈ 5 V).** Command OFF: `sudo pinctrl set 35 op dl`. **Record: blade voltage (expect 0 V).**

2.2.9 Set PSU #3 to −2.50 V and repeat 2.2.8. **Record: ON/OFF voltages.** Return the offset to 0 V and restore the −PS2 GND jumper.

2.2.10 **Off-state leakage.** PS2 commanded OFF. Place a 100 Ω resistor across the +PS2 / −PS2 machine contacts on the docked Scaffold. Measure the voltage across the resistor with the DMM. **Record: voltage in mV (≤ 100 mV = ≤ 1 mA leakage).** Remove the resistor and confirm the floating node bleeds down within a few seconds (100 kΩ bleed) — **record: bleed time in s.**

2.2.11 **Thermal.** Command ON, load 9 A, run 30 minutes. Thermal-image U7 and the PS2 shunt. **Record: final temperature of each in °C.**

PASS: switches at ±2.5 V offset; total drop at 10 A ≤ 100 mV; U7 ≤ 70 °C at 9 A / 25 °C ambient.

2.3 Power meters 1–4 (INA228)

2.3.1 **I2C presence.** Run `sudo i2cdetect -y 1`. **Record: devices shown at 0x40 (PS4), 0x41 (PS3), 0x44 (PS2), 0x45 (PS1) — all four must ACK.**

2.3.2 **Zero offset.** With no external loads, sample each channel once per second for 100 samples: `sudo /opt/cpw-test/power-meter.py --rail <PS1|PS2|PS3|PS4> --samples 100 --interval 1`. **Record per channel: mean in µA and RMS noise in µA (expect |mean| below the µA-range floor for the shunt).**

2.3.3 **Calibration sweep** (PS1, PS2, PS4; PS3 excluded until its shunt is fitted). For each rail set the electronic load to 0.01 / 0.1 / 1 / 5 / 10 A (PS1 also 12 A). At each point **record three numbers: load setpoint in A, DMM-measured current in A, and the reading from `sudo /opt/cpw-test/power-meter.py --rail <rail>` in A.**

2.3.4 Compute and **record the percent error at each point** ((INA − DMM) / DMM × 100).

2.3.5 **Bus voltage.** For PS1: set PSU #2 to 12, 24, 48, then 60 V with no load. **Record at each point: INA bus-voltage reading in V (`power-meter.py --rail PS1 --bus`) and DMM reading at J1 in V.** Verify the other channels at their nominal voltages the same way.

2.3.6 **Thermal validation — setup.** Set the electronic load on PS1 to 12.0 A. Point the thermal camera at the PS1 shunt.

2.3.7 Log the PS1 current once per minute for 30 minutes:
`sudo /opt/cpw-test/power-meter.py --rail PS1 --samples 30 --interval 60`. Note the peak shunt temperature during the soak.

2.3.8 **Record: reading drift over the soak in % ((max − min) / mean × 100, expect ≤ ±0.2 %) and peak shunt temperature in °C.** Set the load to 0 A.

PASS: all four ACK; error ≤ ±0.5 % above 100 mA; 60 V point within ±0.5 %; thermal drift within ±0.2 %.

---

3. CM5 BUSES

3.1 I2C (0–2)

3.1.1 **Edges.** Generate continuous traffic on each bus while scoping — run this loop in a second console (replace `<N>` with 1, 2, then 10). Stop each test with Ctrl+C.

```
while true; do sudo i2cdetect -y <N> > /dev/null; sleep 0.2; done
```

3.1.2 Scope SDA, then SCL, at the pull-up end of each bus: i2c-1 (at the INA228 cluster), i2c-2 (at the sensor slots, both molecules fitted), i2c-10 (at the TCA6424A). **Record per bus: rise time 30—70 % in ns** (spec: ≤ 1000 ns at 100 kHz, ≤ 300 ns at 400 kHz).

3.1.3 **Recognition.** Run `sudo i2cdetect -y 1` — **record: 0x40 / 0x41 / 0x44 / 0x45 present.** Run `sudo i2cdetect -y 2` — **record: 0x76 and 0x68 present (fitted sensor molecules).**

PASS: rise times in spec on all three buses; address maps exactly as designed.

4. ADOM LEDS

4.1 LED ring

4.1.1 Run `sudo i2cdetect -y 10`. **Record: 0x22 ACKs and no unexpected addresses.**

4.1.2 Run the LED test script: `sudo /opt/cpw-test/led-ring.py`. It chases each of the 18 ring LEDs in sequence, then fires each of the 4 status LEDs. **Record: LEDs correct / 22 (any skipped, stuck, or visibly dim channel is a FAIL).**

PASS: 0x22 ACKs; 22/22 LEDs fire.

---

5. USB (SS)

GND jumpers must be in place (§0.2). PS2 must be ON (§2.2) — VBUS derives from it.

5.1 USB-C 3.0 - PD-CNTRL

5.1.1 **ID-gate.** Attach the USB-C sink tester to J10. **Record: the tester detects a source and the CM5 remains host (`lsusb` still lists downstream devices).**

5.1.2 **Measure: power delay.** Scope CH1 on a CC line at the tester breakout, CH2 on VBUS at J10; single-shot on CH1. Plug in and capture. **Record: time from CC attach to VBUS rise in ms, and confirm a soft-start ramp with no overshoot.**

5.1.3 **Measure: advertised current.** If the sink tester has a display, read the advertised current directly from it. Otherwise measure the DC voltage on the active CC line to GND at the breakout with the DMM and read the advertisement from the voltage band: 0.36—0.46 V = default USB; 0.86—1.06 V = 1.5 A; 1.51—1.85 V = 3.0 A. **Record: advertised current (expect the 3.0 A band).**

5.1.4 **Measure: loaded VBUS.** Load VBUS with the electronic load at 0.5 / 1.5 / 3.0 A. **Record: VBUS voltage in V at each point (≥ 4.75 V at 3.0 A).**

5.1.5 **Plug direction.** The plug orientation decides which CC pin is active: the active CC sits at the Rp/Rd divider voltage (≈ 1.5—1.8 V for a 3 A source) and the unused CC stays near 0 V; flipping the plug swaps the two pins. Plug the SSD in orientation A and confirm it enumerates (`lsusb`). Measure CC1 → GND and CC2 → GND with the DMM. **Record: both voltages in V.** Unplug, flip the plug 180°, re-plug. **Record: enumeration again, and CC1/CC2 in V (values swapped vs orientation A).**

PASS: 3.0 A advertisement; VBUS in spec at 3 A; both orientations enumerate.

5.2 USB-C 3.0 - DATA

5.2.1 **SS speed.** Run `lsusb -t` — the SSD's line must end in `5000M`. **Record: negotiated speed.**

5.2.2 **Throughput soak.** Identify the SSD device node with `lsblk` (e.g. `sda`), then read 50 GB from it:

```
sudo dd if=/dev/sda of=/dev/null bs=1M count=51200 status=progress
```

Record: the rate from the last line in MB/s. Then run `sudo dmesg | grep -iE "usb.*reset|disconnect"` — **record: reset/disconnect count (expect 0).**

5.2.3 Repeat 5.2.1—5.2.2 with the plug flipped. **Record: both again.**

5.2.4 **2.0 speed.** Plug the USB2-only flash drive into J10. `lsusb -t` shows `480M`. Transfer 10 GB as in 5.2.2 (adjust `count=10240`). **Record: speed line, rate in MB/s, reset count.**

PASS: zero resets; throughput consistent in both orientations.

5.3 USB-A

5.3.1 **Hot-plug.** Insert and remove the SSD 10 times, waiting for enumeration each time (`lsusb` shows it). Scope VBUS during insertions. **Record: successful enumerations / 10, and the lowest VBUS voltage seen in V (≥ 4.75 V).**

5.3.2 **Measure: spin-up droop.** Scope VBUS at the USB-A connector: 10× probe, AC-coupled, 100 mV/div, 20 ms/div, single-shot, trigger falling at −100 mV. Insert the SSD once and capture. **Record: maximum droop in mV (≤ 330 mV).**

PASS: 10/10 hot-plugs; droop in spec.

5.4 USB power switch (eFuse)

5.4.1 **Float / off state.** Command PS2 OFF (§2.2.2). **Record: VBUS at both USB connectors in V (expect 0 V).** Plug a powered hub into USB-A. **Record: voltage on the board VBUS net in V (expect 0 V — reverse blocking).**

5.4.2 **Measure: ILIM.** PS2 ON. Connect the electronic load to VBUS through a sacrificed USB cable. Ramp the load slowly from 3.0 toward 5.0 A. **Record: the current in A at which the eFuse limits or folds back (design 4.45 A).**

5.4.3 **Measure: retry behavior.** Hold the overload for 5 s and watch VBUS on the scope. Note which fault mode the switch shows: latched (VBUS stays at 0 V until PS2 is cycled) or auto-retry / hiccup (VBUS pulses periodically as the switch retries). Reduce the load to 1 A. **Record: fault mode, and the time from overload removal to VBUS back in regulation in s (expect < 1 s for auto-retry).**

5.4.4 **Fault LED.** While the 5.4.2 overload is applied, **record: red fault LED D23 lit during fault and off after recovery, yes/no.**

5.4.5 **Measure: UVLO / OVP.** With a 1 A load on VBUS, sweep PSU #2 (PS2 input) slowly from 5.0 V down until VBUS cuts out — **record the cut-out voltage in V (design 4.29 V; do not go below 3.5 V — no cut-out by 3.5 V is a FAIL).** Return to 5.0 V, then sweep up until VBUS cuts out — **record the cut-out voltage in V (design 5.68 V; do not exceed 6.0 V — no cut-out by 6.0 V is a FAIL).** Return PSU #2 to 5.00 V.

PASS: thresholds within ±5 % of design values; clean recovery; no back-feed. (Destructive short-circuit test: Appendix A.1.)

---

6. PCIE (WIFI / BT)

6.1 3V3_PCIE (U14)

6.1.1 **Enable.** With no M.2 module fitted: measure the M.2 socket 3.3 V pins with the DMM. **Record: voltage (expect 0 V — the rail stays off until the CM5 asserts PCIE_PWR_EN).** No manual action enables this rail: the CM5 asserts PCIE_PWR_EN by itself during PCIe bring-up at boot — see `pcie-power-explainer.txt`. Power off, fit the AX210 module with both antennas, power on, and measure again. **Record: voltage (expect 3.30 V ± 2 %).**

6.1.2 **Measure: startup ramp.** Scope the M.2 3.3 V pins (single-shot, DC, 1 V/div, 2 ms/div, trigger rising 1.0 V) and power-cycle. **Record: ramp monotonic yes/no, overshoot in mV (≤ 5 %).**

6.1.3 **Ripple under TX — driver install.** Install the Wi-Fi firmware package: `sudo apt update` then `sudo apt install -y firmware-iwlwifi`, then `sudo reboot`.

6.1.4 Confirm the driver loaded: `sudo dmesg | grep iwlwifi` shows a firmware version line with no errors, and `ip link` lists `wlan0`. **Record: driver loaded yes/no.**

6.1.5 Connect to the bench AP: `sudo nmcli dev wifi connect "<SSID>" password "<password>"`. **Record: IP address obtained.**

6.1.6 Generate continuous transmission — run against the link-partner iperf3 server (§1.5) over the Wi-Fi link: `iperf3 -c <partner-ip> -u -b 100M -t 120`.

6.1.7 **Measure: ripple under TX.** While 6.1.6 runs, scope the M.2 3.3 V pins AC-coupled (10× probe, 20 mV/div). **Record: ripple in mV p-p and worst droop in mV.**

PASS: true 0 V off; clean ramp; droop ≤ 3 % during TX bursts.

6.2 WiFi

6.2.1 Module fitted with both antennas (per 6.1.1), driver installed (per 6.1.3). Power on.

6.2.2 **Module detect.** Run `lspci` — the Intel Wi-Fi device appears. Then `sudo lspci -vv | grep -A2 LnkSta` — **record: the link line (expect `Speed 5GT/s, Width x1`).**

6.2.3 **AER baseline.** Run `sudo lspci -vv | grep -iA3 "Advanced Error"` and **record the correctable/uncorrectable counts as the baseline.**

6.2.4 **Firmware.** Run `sudo dmesg | grep iwlwifi`. **Record: firmware version line present, no error lines.**

6.2.5 **Interface.** Run `ip link`. **Record: wlan0 present.**

6.2.6 **Scan / antenna check.** Run `nmcli dev wifi list`. **Record: expected SSIDs visible and the RSSI of a reference AP in dBm.** Power off, remove one antenna, boot, rescan: **record the RSSI drop in dB (a clear drop proves the antenna path).** Refit the antenna.

6.2.7 **Associate + throughput.** Run `sudo nmcli dev wifi connect "<SSID>" password "<password>"` — **record: IP address obtained.** Run iperf3 to the wired partner for 15 min each direction (`iperf3 -c <partner-ip> -t 900`, then add `-R`). **Record: both throughputs in Mbit/s.**

6.2.8 **AER re-read.** Run `sudo lspci -vv | grep -iA3 "Advanced Error"` again. Compare each correctable (CESta) and uncorrectable (UESta) line against the 6.2.3 baseline. **Record: the delta per counter (expect 0 on every counter; any uncorrectable growth is a FAIL, correctable growth gets an engineering note).**

PASS: Gen2 x1 link; association stable; zero AER growth over the soak.

6.3 BT

6.3.1 **Mux to BT — select.** Run `sudo pinctrl set 27 op dh` (mux select line).

6.3.2 **Mux to BT — enable.** Run `sudo pinctrl set 17 op dh` (mux enable line). Confirm the polarity against the schematic on the first run and note the working levels on the sheet.

6.3.3 Run `lsusb`. **Record: Intel Bluetooth controller listed.**

6.3.4 **Scan / pair.** Run `bluetoothctl`, then inside it: `scan on`. **Record: a known device appears; `pair <MAC>` succeeds.**

6.3.5 **Stability.** Run `sudo l2ping -c 900 <MAC>` (≈15 min). **Record: packet loss % and `sudo dmesg | grep -i hci` reset count (expect 0).**

6.3.6 **Exclusivity — USB2 device.** With BT active, plug the USB2-only flash drive into USB-A. Run `lsusb`. **Record: the drive does NOT appear.**

6.3.7 **Exclusivity — SS device.** Plug the USB3 SSD into USB-A. Run `lsusb -t`. **Record: the SSD's line ends in `5000M` (SuperSpeed is unaffected by the mux).**

6.3.8 **Mux glitch — start a SS transfer.** Plug the SSD into the USB-C (J10), find its node with `lsblk`, then start a ≈20 GB background read:

```
sudo dd if=/dev/sda of=/dev/null bs=1M count=20480 status=progress &
```

6.3.9 While the transfer runs, toggle the mux 10 times, ≈1 s apart, alternating: `sudo pinctrl set 27 op dl` then `sudo pinctrl set 27 op dh`.

6.3.10 When the transfer finishes, run `sudo dmesg | grep -iE "usb.*reset|disconnect"`. **Record: reset/disconnect count (expect 0).**

PASS: all steps as stated; exclusivity behavior documented on the sheet.

7. CAN

Setup: partner node on twisted pair at J3 (CANH/CANL, either WAGO contact pair). JP6 as noted per step. Install the CAN tools on the CM5 first: `sudo apt install -y can-utils`.

7.1 Controller communications (SPI)

7.1.1 **Install the device-tree overlay** (one-time per image). Append the MCP251863 overlay line to the boot configuration, then reboot:

```
echo 'dtoverlay=mcp251xfd,spi0-0,oscillator=40000000,interrupt=5' | \
  sudo tee -a /boot/firmware/config.txt
sudo reboot
```

7.1.2 **Probe check.** Run `sudo dmesg | grep -i mcp251` — a successful probe names the device — and `ip link show can0` — the interface exists. **Record: probe result.**

7.1.3 (only if 7.1.2 fails) Scope the U15 SDO pin during a probe attempt (`sudo reboot` and watch during boot): DC, 1 V/div, 10 µs/div. Toggling means the controller is answering; a flat line means no response. **Record: SDO toggling yes/no.**

7.1.4 (only if 7.1.2 fails) Check with the DMM, one item at a time: 3v3 at U15 VDD; nRST high (> 3.0 V); nCS strobing during a probe attempt (scope, single-shot falling); SCK activity. **Record: each result.** Route failures to engineering.

7.1.5 **SPI clock sweep — set 1 MHz.** Edit the overlay line (`sudo nano /boot/firmware/config.txt`) to read `dtoverlay=mcp251xfd,spi0-0,oscillator=40000000,interrupt=5,spimaxfrequency=1000000` , then `sudo reboot` .

7.1.6 Exercise register access with 1000 loopback frames (no bus partner involved):

```
sudo ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on loopback on
candump can0 > $HOME/spifreq.log &
cangen can0 -g 2 -I i -L 8 -n 1000
wc -l $HOME/spifreq.log
```

Record: frames received / 1000. Then clean up: `sudo ip link set can0 down` and `rm $HOME/spifreq.log` .

7.1.7 Repeat 7.1.5—7.1.6 with `spimaxfrequency` set to 5000000, 10000000, 15000000, then 20000000. **Record: the highest frequency giving 1000/1000, in MHz.**

7.1.8 **Set the production value.** Edit `spimaxfrequency` to ≤ 70 % of the 7.1.7 result and `sudo reboot` . **Record: the production frequency in MHz.**

PASS: driver probes; stable register access at the derated frequency.

7.2 Internal loopback

7.2.1 Bring the interface up in loopback mode:

```
sudo ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on loopback on
```

7.2.2 Start logging received frames in the background: `candump can0 > $HOME/loopback.log &`

7.2.3 Send 1000 mixed frames: `cangen can0 -g 2 -I i -L 8 -n 1000`

7.2.4 Count the received frames: `wc -l $HOME/loopback.log` . **Record: frames received / 1000.** Stop the logger (`kill %1`) and delete the log (`rm $HOME/loopback.log`).

7.2.5 Run `ip -details -s link show can0` . **Record: TX/RX error counters (expect 0).**

PASS: 1000/1000, zero errors.

7.3 Bus levels / termination (JP6)

7.3.1 **Bus levels.** JP6 OPEN, partner connected but idle. Bring the link up and transmit:

```
sudo ip link set can0 down
sudo ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on
cangen can0 -g 10
```

Scope CANH–CANL differentially at J3 while it transmits. **Record: dominant differential voltage in V (≈ 2.0) and recessive in V (≈ 0).** Stop cangen with Ctrl+C.

7.3.2 **Termination.** JP6 CLOSED. Power the board OFF and disconnect the partner cable so nothing else loads the bus. Set the DMM to resistance and measure across CANH and CANL at J3. **Record: resistance in Ω — expect ≈ 120 Ω, the two 60 Ω halves of the split termination in series (the center capacitor is invisible to a DC measurement).**

7.3.3 **Restore.** Set JP6 to the deployment configuration for this unit and note which state that is. Power on, reconnect the partner, bring the link up (7.3.1 syntax), and run `candump can0` — partner frames appear within a few seconds. **Record: JP6 state used, and partner frames seen yes/no.**

PASS: levels and termination as designed.

7.4 FD interop

7.4.1 Bring the link up against the partner (loopback OFF):

```
sudo ip link set can0 down
sudo ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on
```

Record: link up, partner frames visible in `candump can0`.

7.4.2 Run 1,000,000 frames: `cangen can0 -g 0.5 -b -I i -L 8 -n 1000000` while the partner echoes or logs. Afterward run `ip -details -s link show can0`. **Record: TX/RX error counters and bus-off count (expect all 0).**

7.4.3 Repeat at a 5 Mbit/s data rate (short bus, JP6 per deployment, partner set to 5 M as well) — run each line:

```
sudo ip link set can0 down
sudo ip link set can0 up type can bitrate 500000 dbitrate 5000000 fd on
cangen can0 -g 0.5 -b -I i -L 8 -n 1000000
ip -details -s link show can0
```

Record: TX/RX error counters and bus-off count (expect all 0).

PASS: 1 M frames clean at 2 M; 5 M functional. (Destructive transceiver fault test: Appendix A.2.)

7.5 Interrupts

7.5.1 **Create RX load.** On the partner node (if it runs Linux + can-utils) start a continuous stream toward the board: `cangen can0 -g 1 -I 5A5 -L 8` (≈1000 frames/s). Any partner that can stream ≥ 100 frames/s continuously works — record what was used.

7.5.2 On the CM5 run `watch -n1 'grep -i mcp /proc/interrupts'`. **Record: the interrupt count increments while the partner streams and stops when the partner stops (Ctrl+C on the partner) — no storming, no stuck line.**

PASS: counts track traffic.

---

8. NESTED

8.1 PiProbe

8.1.1 **Connection check.** Run `lsusb` on the CM5. **Record: a CMSIS-DAP device is listed** (the probe, on the CM5's dedicated USB2 port).

8.1.2 **Flashing (the probe itself) — enter bootloader.** Run the four lines in order:

```
sudo pinctrl set 22 op dl   # hold probe BOOTSEL low
sudo pinctrl set 23 op dl   # assert probe reset
sudo pinctrl set 23 op dh   # release reset (BOOTSEL still low)
sudo pinctrl set 22 op dh   # release BOOTSEL
```

8.1.3 Run `lsblk`. **Record: a new UF2 mass-storage device appears** (note its node, e.g. `sda1`).

8.1.4 Copy the probe firmware onto it (adjust the node from 8.1.3):

```
sudo mount /dev/sda1 /mnt
sudo cp /opt/cpw-test/fw/piprobe.uf2 /mnt/
sudo umount /mnt
```

The probe reboots itself when the copy completes.

8.1.5 Run `lsusb` again. **Record: the CMSIS-DAP device re-enumerated.**

PASS: stable enumeration; full self-recovery flash cycle.

8.2 MIC

8.2.1 **Connection check.** Find the capture device: `arecord -l` (note the card/device numbers, e.g. card 1 device 0). Start a capture to keep the clocks running:

```
arecord -D plughw:1,0 -f S32_LE -r 48000 -c 2 /dev/null
```

Scope BCLK and LRCLK at the N3 pads. **Record: LRCLK frequency in kHz (expect 48.0) and BCLK frequency in MHz (expect 3.072 = 48 kHz × 64).** Edge quality expectations: swing 0 V ↔ 3.3 V, monotonic edges, overshoot/undershoot ≤ 330 mV (10 %), no runt pulses or double transitions. **Record: edges within expectations yes/no.** Stop the capture with Ctrl+C.

8.2.2 **Measure: noise floor.** Install the analysis tool once: `sudo apt install -y sox`. In a quiet room capture 60 s: `arecord -D plughw:1,0 -f S32_LE -r 48000 -c 2 -d 60 quiet.wav`, then analyze: `sox quiet.wav -n stats`. **Record: RMS level in dBFS, and confirm the signal is in the left channel (SEL = GND) with the right channel silent/mirrored.**

8.2.3 **Measure: tone capture.** Play a 1 kHz tone from a phone or speaker ≈ 30 cm from the mic and capture 10 s while it plays, then analyze:

```
arecord -D plughw:1,0 -f S32_LE -r 48000 -c 2 -d 10 tone.wav
sox tone.wav -n stats
sox tone.wav -n stat 2>&1 | grep "Rough frequency"
```

Record: tone RMS level in dBFS (from `stats`), its margin above the 8.2.2 floor in dB, and the rough frequency (expect ≈ 1000).

8.2.4 **EMI check.** Start network load from the link partner (`iperf3 -c <partner-ip> -t 60`) and the LED chase (`sudo /opt/cpw-test/led-ring.py --loop 10`). While both run, capture and analyze:

```
arecord -D plughw:1,0 -f S32_LE -r 48000 -c 2 -d 30 emi.wav
sox emi.wav -n stats
```

Record: RMS level in dBFS vs the 8.2.2 floor (expect within 1 dB — no new tones or artifacts).

PASS: floor recorded; 1 kHz clean; no EMI artifacts.

8.3 Sensor

8.3.1 **I2C connection.** Both molecules fitted: `sudo i2cdetect -y 2` . **Record: 0x76 (BME690) and 0x68 (BMI270) ACK.**

8.3.2 **BME690 chip ID.** Run `sudo i2cget -y 2 0x76 0xd0` . **Record: returned value (expect 0x61).**

8.3.3 **BMI270 chip ID.** Run `sudo i2cget -y 2 0x68 0x00` . **Record: returned value (expect 0x24).**

8.3.4 **Interrupt.** Only the BMI270 has an interrupt line (the BME690 has no INT pin). Configure a 100 Hz data-ready interrupt: `sudo /opt/cpw-test/sensor-check.py --imu-int --rate 100` , and scope GPIO16. **Record: interrupt pulses at the configured 100 Hz, and no contention on the shared open-drain line.**

8.3.5 **Environmental data.** Run `sudo /opt/cpw-test/sensor-check.py --env` and compare against a room reference. **Record: temperature (±2 °C), humidity (±5 %RH), pressure (±5 hPa).**

8.3.6 **IMU data.** Run `sudo /opt/cpw-test/sensor-check.py --imu` with the board flat. **Record: Z-axis ≈ +1 g.** Flip the board upside down and re-run. **Record: Z-axis ≈ −1 g.**

PASS: both IDs correct; interrupt tracks the configured rate; all reads sane.

---

9. SCAFFOLD

The Scaffold board is already docked (§0.2). Configure its jumpers per its own documentation (single ground bond per rail rule). If you are testing only a Scaffold board, start here (§0 setup still applies) — the workcell host must have passed §1—§2, §5, and §8.1 at minimum.

9.1 Power

9.1.1 **Measure: 3V3 ramp at the MCU.** Probe the RP2350's 3v3 at its decoupling capacitors: single-shot, DC, 1 V/div, 2 ms/div, trigger rising 1.0 V. Power-cycle the workcell and capture the ramp at dock power-on. **Record: monotonic yes/no, settled voltage in V (3.30 ± 2 %).**

9.1.2 Flash the IO-stress firmware from the CM5 (cpw-walk1 toggles all 26 USER_IO continuously):

```
sudo openocd -f interface/cmsis-dap.cfg -f target/rp2350.cfg \
  -c "adapter speed 1000" \
  -c "program /opt/cpw-test/fw/cpw-walk1.elf verify reset exit"
```

If openocd is not installed yet: `sudo apt install -y openocd` .

9.1.3 **Measure: 3V3 under IO load.** With cpw-walk1 running, re-probe the 3v3: AC-coupled, 10× probe, 20 mV/div. **Record: ripple in mV p-p and worst droop in mV.**

9.1.4 **Measure: Scaffold eFuse ILIM.** Electronic load on the Scaffold's USB VBUS (at the VBUS machine contact). PS2 ON. Ramp the load slowly from 1.0 A upward until the eFuse trips. **Stop at 4.5 A maximum — no trip by 4.5 A is a FAIL. Record: trip current in A, and the Scaffold fault LED lit during the fault / clear after.**

9.1.5 **Measure: +PS2 delivered voltage.** Set the electronic load to 5.0 A across the +PS2 / −PS2 machine contacts on the Scaffold. **Record: voltage at the contacts in V and the delta vs the carrier-side blade P2 in mV** (connector drop budget). Set the load to 0 A.

PASS: MCU rail clean idle and under IO load; eFuse trips at ≤ 4.5 A; contact drop recorded.

9.2 USB

9.2.1 **Client role — enumeration.** Cable the Scaffold USB-C to the link-partner PC. With a blank RP2350, run `lsusb` on the PC. **Record: `Raspberry Pi RP2 Boot` listed** (BOOTSEL mass-storage). If application firmware is already loaded, **record its CDC device instead.**

9.2.2 **Client role — no source contention.** With the PC attached, measure VBUS at the Scaffold USB-C with the DMM. **Record: VBUS in V (≈ 5 V, supplied by the PC) and enumeration stable — in client role the Scaffold must not source VBUS (the eFuse blocks back-feed into PS2).**

9.2.3 **Host role — VBUS sourcing.** Unplug the PC. PS2 ON. Plug the USB2-only flash drive into the Scaffold USB-C — the drive's pull-downs put the TUSB321 in host role, Q1 turns on, and the board sources VBUS from +PS2. Measure VBUS at the connector with the DMM. **Record: VBUS in V (expect ≈ 5.0, sourced by the Scaffold).**

9.2.4 **Host role — enumeration.** Flash the host-test firmware (`/opt/cpw-test/fw/cpw-usbhost.elf` , over SWD as in 9.1.2). Re-plug the flash drive. The firmware reports detected devices on the debug UART (open it per 9.3.8—9.3.9). **Record: the drive is reported, yes/no.**

9.2.5 **Measure: client-role speed.** Return the RP2350 to BOOTSEL (9.3.13) with the PC connected. On the PC, create a 16 MB file and time the copy onto the `RP2350` drive:

```
dd if=/dev/zero of=test.bin bs=1M count=16
time cp test.bin /media/$USER/RP2350/
```

Record: MB/s (16 ÷ seconds). The RP2350 port is USB 2.0 Full-Speed — expect ≈ 0.5—1 MB/s.

PASS: both roles functional; no back-feed in client role; speed recorded.

9.3 MCU (RP2350)

9.3.1 Install the tools once on the CM5: `sudo apt install -y openocd picotool picocom` .

9.3.2 **SWD scan.** Run:

```
sudo openocd -f interface/cmsis-dap.cfg -f target/rp2350.cfg \
  -c "adapter speed 1000" -c "init" -c "targets" -c "shutdown"
```

Record: the RP2350 IDCODE reported.

9.3.3 Re-run 9.3.2 raising `adapter speed` to 2000, 5000, 10000, then 15000. **Record: the highest speed in kHz at which the IDCODE still reads.**

9.3.4 **Record: the production SWD speed = ≤ 70 % of the 9.3.3 result, in kHz.** Use it in every flash step below.

9.3.5 **Flash + verify.** Program the heartbeat test image:

```
sudo openocd -f interface/cmsis-dap.cfg -f target/rp2350.cfg \
  -c "adapter speed <production speed>" \
  -c "program /opt/cpw-test/fw/cpw-heartbeat.elf verify reset exit"
```

Record: `Verified OK` printed.

9.3.6 **Record: the heartbeat running** — cpw-heartbeat blinks the Scaffold status LED at 1 Hz and prints an incrementing counter on the debug UART.

9.3.7 **UART echo — flash.** Program the echo firmware:

```
sudo openocd -f interface/cmsis-dap.cfg -f target/rp2350.cfg \
  -c "adapter speed <production speed>" \
  -c "program /opt/cpw-test/fw/cpw-echo.elf verify reset exit"
```

9.3.8 Find the probe's CDC UART: `ls /dev/ttyACM*` (expect `/dev/ttyACM0`).

9.3.9 Open it at 115200 baud: `picocom -b 115200 /dev/ttyACM0` . Type `THE QUICK BROWN FOX 0123456789` — every character echoes back as typed. **Record: echo intact yes/no.** Exit with Ctrl+A then Ctrl+X.

9.3.10 Repeat at 1 Mbaud: `picocom -b 1000000 /dev/ttyACM0` . **Record: echo intact yes/no.**

9.3.11 Repeat at 3 Mbaud: `picocom -b 3000000 /dev/ttyACM0` . **Record: echo intact yes/no.**

9.3.12 (only if silent at every rate) Check continuity of the TX→RX crossover at molecule pad 32. **Record: the finding.**

9.3.13 **Boot / reset control.** Drive the RP2350 into BOOTSEL from the CM5 — run the four lines in order:

```
 sudo pinctrl set 34 op dl   # assert MCU_BOOT (CAM_GPIO0)
sudo pinctrl set 21 op dl   # assert MCU_RST - RP2350 held in reset
sudo pinctrl set 21 op dh   # release reset with BOOT held: enters bootloader
sudo pinctrl set 34 op dh   # release BOOT
```

9.3.14 Verify on the PC attached to the Scaffold USB-C (the RP2350's USB routes out the Scaffold connector, not to the CM5): `lsusb` on the PC shows `RP2 Boot` and an `RP2350` drive appears. **Record: BOOTSEL entered yes/no.**

9.3.15 Repeat 9.3.13—9.3.14 twenty times. Between cycles, return to the application with a plain reset: `sudo pinctrl set 21 op dl` then `sudo pinctrl set 21 op dh` . **Record: successes / 20.**

9.3.16 **UF2 flash.** With the RP2350 in BOOTSEL, drag `cpw-heartbeat.uf2` (from `/opt/cpw-test/fw/` , copied to the PC) onto the `RP2350` drive. **Record: the drive disconnects and the heartbeat (9.3.6) resumes.**

9.3.17 **USER_IO — flash walking-1.** Program the walking-1 firmware (each of the 26 USER_IO lines pulses high in sequence, 100 ms per line, endlessly):

```
sudo openocd -f interface/cmsis-dap.cfg -f target/rp2350.cfg \
  -c "adapter speed <production speed>" \
  -c "program /opt/cpw-test/fw/cpw-walk1.elf verify reset exit"
```

9.3.18 Verify each line at the Scaffold machine contacts with the scope or DMM (or the loopback jig). **Record: lines verified / 26.**

9.3.19 Scope each of the four reserved GPIO (10 / 11 / 20 / 21) for ≥ 5 s each. **Record: they never toggle.**

PASS: SWD margin recorded; 20/20 boot-control cycles; 26/26 IO verified, reserved pins untouched.

APPENDIX A — ENGINEERING-ONLY TESTS ⚠

Potentially destructive. Run only on a designated engineering (sacrificial) board — never on a deliverable unit.

A.1 USB VBUS crowbar (companion to §5.4)

A.1.1 PS2 ON, VBUS live. Momentarily short VBUS to GND with a wire. **Record: the eFuse catches it (VBUS collapses, no PS2 PSU trip, power meter 0x44 shows the event) and VBUS recovers after removal.**

A.2 CAN transceiver fault tolerance (companion to §7)

A.2.1 With traffic attempted (`cangen can0 -g 10`), apply each fault for 1 minute, then remove it and confirm traffic resumes: (a) CANH shorted to CANL; (b) CANH to +12 V; (c) CANL to GND. **Record: recovery after each fault, yes/no ×3.**

End of procedure. Record all values on the Control Panel Test Method Overview sheet; attach scope captures referenced by section number.