



Quick signal and power integrity checks for KiCad Documentation

Version 0.3.0 · 2026-10-25 · commit 092a693

<https://github.com/riehlies/kisi> · live version: riehlies.github.io/kisi

kisi is a quick design check: its results are estimates, without any warranty of correctness – not a replacement for sign-off simulation or measurement. The documentation is part of kisi, GPL-3.0-only. Rendered from `docs/` by `tools/docs_pdf.py`; the pages on GitHub are the same text.

Contents

Getting started	4
What you need	4
The files of a release	4
The viewer	4
The command line	8
Next steps	8
Preparing a board	9
The workflow	9
The gap list	9
Four ways to fill it in	11
Worked example: Seeed Studio XIAO nRF54LM20A	12
Field and kisi.json reference	16
What kisi reads from the KiCad project	16
Footprint fields	17
Board configuration <board>.kisi.json	18
Other input files	20
Output files	21
Models and limits	22
Scope: what kisi computes and what it leaves to other tools	22
Why kisi is small and fast	22
Impedance	22
Lines	23
Vias	23
Lanes and the channel check	24
Crosstalk	24
Power	24
Temperature (estimate)	24
Channel export	25
Accuracy levels	25
What is an estimate	26
Validation	26
Limits	26
CLI reference	28
kisi help	28
Commands	29
Options	31
Assembly variants	32
Field solver	33
Threads	33
Exit codes	33

Using kisi with agents	34
The skills	34
Setup	34
Notes for agents	34

Links in this PDF lead to the pages and headings; links to other files open them on GitHub.

Getting started

kisi reads a KiCad layout and checks trace impedance, high-speed channels and power delivery. It runs in the browser (the viewer) or on the command line; both use the same engine and give the same results.

What you need

- The board file `.kicad_pcb` (KiCad 8, 9 or 10), saved.
- The project file `.kicad_pro` next to it, for the net classes (target impedances and interface profiles come from them).
- Optionally `<board>.kisi.json`: the values the board file lacks (laminates data, load currents, ...) with their sources – see [Preparing a board](#). Assembly variants may name BOM files, connector ends Touchstone files; drop or keep them next to it.

The files of a release

Every [release](#) carries:

File	What it is
<code>kisi.html</code>	the viewer: one HTML file that runs offline in any current browser
<code>kisi-examples.html</code>	the same viewer with two example boards built in – ThunderScope Rev5.3 and LibreVNA – each with its power tree as a <code>kisi.json</code>
<code>kisi.mjs</code>	the command line for scripts, CI and agents (Node.js ≥ 18), with the 2D field solver
<code>kisi-docs.pdf</code>	this documentation as one PDF
<code>kisi-references.html</code>	3D reference cases for vias, stubs, pairs, lines and plane cavities with field plots, and kisi's fast models against them (attached to the releases in which the cases changed, currently v0.3.0)

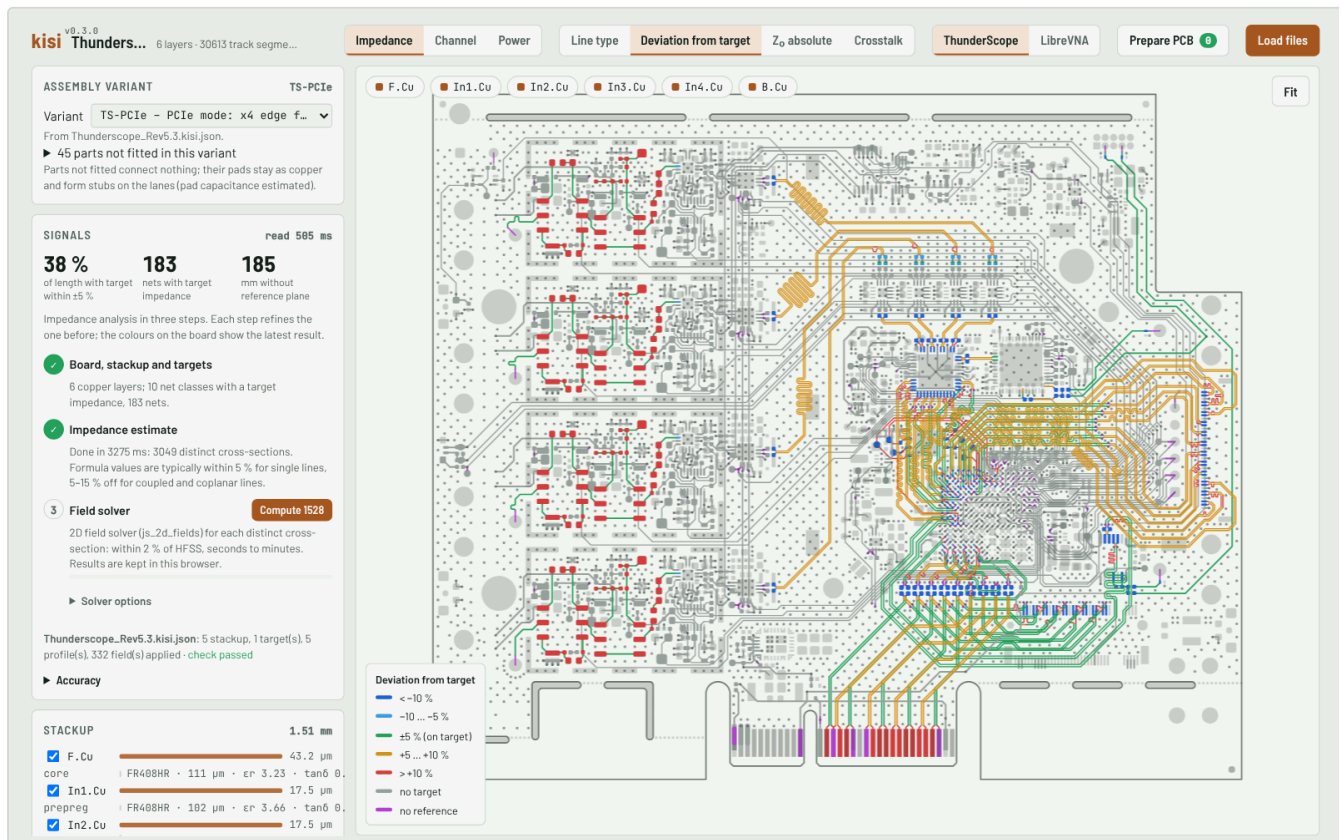
The [live version](#) serves the viewer and the example viewer from GitHub Pages.

The viewer

Open `kisi.html` (or the live version) and drop the `.kicad_pcb` with the `.kicad_pro` – and the board's `kisi.json`, BOMs and Touchstone files if there are any – onto the page, or pick them with the file dialog. Nothing is installed, the page loads nothing from the web, and the board never leaves your computer. In Chromium-based browsers "Reload" reads the same files again after saving in KiCad.

The header switches between three views. Each starts with what the board file gives and computes more on demand, in numbered steps.

1. Impedance



Stackup and target impedances first. "Estimate impedances" computes formula values for every track in a second, then the 2D field solver refines them – for the selected net, a class, all nets with a target, or everything; results are cached in the browser. The map colours every track by its deviation from the target.

Per net: impedance along the path, loss, TDR, vias and crosstalk. The path runs from A to B (marked on the board and under the charts; A is the transmitter side of the lane or the IC, $\rightleftharpoons$ swaps it). Branches are marked where they leave the path and included in loss and TDR as open stubs; hovering a chart shows the position on the board.

The estimate also gives NEXT/FEXT for every net with a target: the Couplings card lists the critical nets (from 1%), the map mode "Crosstalk" marks the critical stretches, and per net a chart shows where along the path the crosstalk comes from. The field solver refines only the critical couplings (on by default).

2. Channel

ASSEMBLY VARIANT TS-PCIe
Variant: TS-PCIe - PCIe mode: x4 edge f...
From ThunderScope_Rev5.3.kisi.json
► 45 parts not fitted in this variant
Parts not fitted connect nothing; their pads stay as copper and form stubs on the lanes (pad capacitance estimated).

CHANNEL CHECK 77 channels
Channel check in four steps (the field solver is optional). The board map and the interfaces overview show the latest result.
✓ **Lanes and profiles**
25 buses: 23 lanes device to open, 22 device to connector, 14 not checked (open, multi-drop trees). Profiles per net class ("Interfaces per net class"). Variant TS-PCIe (ThunderScope_Rev5.3.kisi.json).
✓ **Impedance estimate**
Done: formula values for every track.
3 **Field solver** Compute 1528
2D field solver (js_2d_fields) for each distinct cross-section: within 2 % of HFSS, seconds to minutes. Results are kept in this browser.
► Solver options
✓ **Check of all channels** Recompute
Done: 77 channels (45 lanes, 32 nets without devices); 20 green, 57 yellow, 0 red. Click a row for the channel.

Net	Profile
PCIe_PET0_P (P1-U18)	PCIe 4.0
PCIe_PET1_P (P1-U18)	PCIe 4.0
PCIe_PET2_P (P1-U18)	PCIe 4.0
PCIe_PET3_P (P1-U18)	PCIe 4.0
REFINOUT2 (J3-U7)	Generic single-ended
PCIe_PER1_P (P1-U18)	PCIe 4.0
PCIe_PER3_P (P1-U18)	PCIe 4.0

Interfaces · ThunderScope_Rev5.3
Signals with a target impedance, followed from device to device through series resistors and AC-coupling capacitors. Terminations (||), branches (-) and side stubs (unused pads, parts not fitted) are part of the channel. Profiles come from the net class ("Interfaces per net class" card); for lanes that leave the board at a connector, the card of the interface sets what sits there and the channel behind it. Assembly variant: TS-PCIe (ThunderScope_Rev5.3.kisi.json). 51 nets only connect parts not fitted: not used in this variant, faint on the board map. On the board map, a click on a channel opens it.

25 buses
23 device to device
22 device to connector
14 not checked
20 / 25 / 0 lanes green / yellow / red

Device to device 8 buses, 23 lanes · transmitter and receiver on this board: eye, loss, return loss, skew

• **U10 ↔ U18** LVDS_SYNC · LVDS (1 Gbit/s) · 2 lanes
U10 FIN1019MTCX ↔ U18 IC FPGA XC7A50T-2CSG325C · direction per lane Swap direction Export...

Lane	Path	mm	IL dB	RL dB	Eye %	Skew ps
• SYNC_IN_P pair	U10.12 → U18.D8	84.1	-0.4	26.3	92	2.7
• SYNC_OUT_P pair	U18.D9 → U10.10 R60 100	85.0	-0.4	27.0	89	16.4

• **U12.1 ↔ U6** FE_1002_Diff · Generic differential 100 Ω · 1 lane
U12.1 ADA4930-1YCPZ → U6 HMCAD1520 default order Swap direction Export...

Lane	Path	mm	IL dB	RL dB	Eye %	Skew ps
• OUT_P pair	U12.1.10 → R137.1 4.99 → U6.38	10.5	-0.5	25.4	95	0.6

• **U12.2 ↔ U6** FE_1002_Diff · Generic differential 100 Ω · 1 lane
U12.2 ADA4930-1YCPZ → U6 HMCAD1520 default order Swap direction Export...

Lane	Path	mm	IL dB	RL dB	Eye %	Skew ps
• OUT_P pair	U12.2.10 → R137.2 4.99 → U6.41	7.0	-0.4	25.7	95	0.0

• **U12.3 ↔ U6** FE_1002_Diff · Generic differential 100 Ω · 1 lane
U12.3 ADA4930-1YCPZ → U6 HMCAD1520 default order Swap direction Export...

The interfaces overview: buses and device-to-device lanes through series resistors and AC-coupling capacitors, with terminations, branches and side stubs. Each net class gets an interface profile (PCIe, USB, DDR, LVDS, Ethernet, HDMI, MIPI, ... from `data/interfaces.json`) with transmitter, receiver and limits; the "Profile" card shows and edits it. The check judges every lane on loss, return loss, skew, via stubs and a linear eye.

- A **differential pair** is a 4-port: both lines on their own paths with their own parts, stubs and vias, coupled where they run together. Loss and return loss are S_{dd21}/S_{dd11} , the eye drives P and N in opposite phase, and the mode conversion (S_{cd21}) is shown.
- A **multi-drop line** along one path (one transmitter, more devices on side stubs) is checked at every receiver, all loading the line; the worst eye counts.
- **Lanes from a device to a connector** are judged on the board share (loss, return loss, skew). Their eye uses the end set in the interface's card – the termination when the connector receives, the driver when it sends – and, if given, the channel behind the connector as a lossy line or a Touchstone file (`.s2p` , `.s4p`). Without it the eye covers the board share only and is information; with it the eye is judged. Lanes from connector to connector (coupons, pass-through boards) get the board share only, without an eye.
- Parts from a lane's net to ground (a shunt capacitor) are in its channel at their pads.
- **Export...** in the lane detail and in the bus card (once the check has computed the lane) downloads the lane or the whole bus as SPICE subcircuits with test benches, Touchstone files and a KiCad project for circuit and RF simulators; the card "Export for simulators" sets what is written and the frequency range (**Channel export**).

The board map shows the channels of the chosen variant – before the check in one colour, after it by result. Nets that only connect parts not fitted are drawn faint and not checked. A click on a channel opens it.

3. Power



Power tree, budget and power flow (Sankey) first, from `PI_REG / PI_LOAD` fields or the board's `kisi.json`; part references jump to the board. Then, in steps:

1. per rail and for the ground return, voltage drop and current density on every layer, the drop at each load and the vias with the highest currents;
2. a losses map: heat per part and copper loss per layer (mW/mm^2);
3. the temperature (an estimate for an ambient temperature and still air or an air flow): a map per layer, the hottest parts with their junction temperature, the rails' IR drop at temperature;
4. $Z(f)$: a lumped model of regulator, plane and capacitors with a plane model above 1 MHz, and the impedance seen from each IC – the rail domain with ferrites and series parts up to the regulator, the capacitors ranked by their loop inductance to the IC (outlined on the board), anti-resonances, a target from the IC's current step and recommendations (value, package, count, distance).

Assembly variants

Boards with assembly variants (KiCad 10 variants or `variants` in `kisi.json`) – or with forks where all alternative parts are fitted – ask "Which assembly variant?" before lanes are computed; the card "Assembly variant" switches later, and the lane results name the variant. A lane is named after and made of its own path; where a line forks with all alternatives fitted, each lane leaves the IC on its own branch and carries the other path as an open branch (dashed on the board).

Prepare PCB

"Prepare PCB" (between the example boards and "Load files") opens what the board file lacks, per category, with forms for the simple values and for everything a loaded `kisi.json` holds, a draft kept in this browser until it is saved, and the views' own edits taken into the file. It is a checklist: each step says what it needs and shows how many of its items are open, "Apply" checks it with the core and gives it a check mark, and the next one opens; the badge on the button counts the open points. The example boards come with a complete `kisi.json`: their checklist is finished and shows the values in its forms. See [Preparing a board](#).

What the viewer stores

Stackup (ϵ_r , $\tan\delta$, thickness), profiles, the net class → profile assignment, power settings, lane directions and the chosen variant can be edited; edits are stored in this browser per board and lie over the kisi.json (Prepare PCB moves them into the kisi.json). The KiCad files are never changed. Field-solver results are cached in the browser (IndexedDB); the results of the last three boards stay in memory while switching. The "Accuracy" section of the impedance view summarises the model accuracy ([Models and limits](#)).

The command line

```
curl -fsSL0 https://github.com/riehlies/kisi/releases/latest/download/kisi.mjs
node kisi.mjs help
node kisi.mjs gaps      board.kicad_pcb    # what the analyses miss
node kisi.mjs overview board.kicad_pcb    # stackup and impedance per net class
node kisi.mjs check     board.kicad_pcb    # channel check of all interfaces
node kisi.mjs power     board.kicad_pcb    # regulator chain, IR drop, losses
node kisi.mjs export     board.kicad_pcb --lane NET # a lane for SPICE, Touchstone and KiCad
```

Every command prints JSON and states the models and assumptions it used. The first run of a field-solver command on a board can take a few minutes; results are cached. All commands and options: [CLI reference](#).

Next steps

- A board straight from KiCad gives a first picture; for meaningful numbers, fill the gaps: [Preparing a board](#).
- What the numbers mean and how far to trust them: [Models and limits](#).
- Let an agent do the research and the review: [Using kisi with agents](#).

Preparing a board

A board straight from KiCad gives a first picture: copper, stackup, net classes, parts. Many values a meaningful analysis needs are not in the board file, or only as KiCad defaults – the laminate's ϵ_r and $\tan\delta$, the interface standard of a bus, load currents, the regulator chain, capacitor and thermal data. Preparing a board means finding these values, with a source for each, and writing them to `<board>.kisi.json` next to the board. kisi never writes into the KiCad files; CLI and viewer lay the kisi.json over the board.

The work has two halves:

- **deterministic:** `kisi gaps` lists what is missing, where to find it and where it goes in kisi.json, and `kisi config-check` checks what was written;
- **research:** reading datasheets and choosing values – by hand, or by an agent with the [kisi-prepare skill](#).

The workflow

1. **Gap list:** `node kisi.mjs gaps board.kicad_pcb`. Work through `high` first, then `medium`; `low` gaps refine the PDN and the temperature estimate.
2. **Context:** `kisi overview` (stackup with materials, net classes, targets), `kisi interfaces` (the devices at the ends of each bus, with their MPNs), `kisi profiles` (the interface profiles a class can get).
3. **Skeleton:** `kisi config-template board.kicad_pcb > board.kisi.json` gives this board's dielectric indices, net classes and suggested variants; keep only what you fill in.
4. **Fill in** values with sources ([format](#)): typical values for loss and eye, maximum currents for IR drop and regulator loading, unless you have a reason otherwise – and say which in the source's `note`. If a value cannot be found, leave it out or mark it `estimate` / `assumption` with the reasoning.
5. **Check:** `kisi config-check board.kicad_pcb` – fix every error (references, ranges, missing sources, field syntax kisi cannot read) and read the warnings. Then `kisi gaps` again: the filled gaps are gone.
6. **Use it:** every command reads the kisi.json next to the board; in the viewer, drop it together with the board files.

Each gap names the analysis categories it matters for (`views`: impedance, channel, power; where a gap mixes them, each item names its own – a dielectric with only KiCad's default $\tan\delta$ is a channel item), and `kisi gaps` says per category whether there is anything to check and whether it is ready (no open high or medium gap). To prepare a board for one analysis only, use `kisi gaps --for impedance` (or `channel`, `power`): a board without nets with a target impedance has nothing to check for impedance, and its laminate gap blocks nothing yet.

Gaps that are guesses from names (`nets-without-target`, rails found only by their name) can be dismissed with a reason when the name misleads – `gaps.dismissed` in kisi.json ([reference](#)).

Gaps whose `write.to` says "not kisi.json" are fixes in KiCad (a Value field kisi cannot read, for example). Nets without a suitable class (in "Default") get one in kisi.json with `net_classes.<class>.nets` (names or patterns, [reference](#)), or in KiCad.

The gap list

`kisi gaps` prints a list of gaps; each has an `id`, a `priority`, the `issue`, what it `affects`, `where_to_find` the value, where to `write` it (with an example) and the `items` – layers, classes, rails or parts (with their MPN where the Value field is not the part number).

Gap	Priority	Category	What is missing	Where it goes
<code>stackup-missing</code>	high	I, C	no stackup in the board file	<code>stackup.dielectrics</code> (or the stackup in KiCad)
<code>stackup-defaults</code>	medium	I, C	dielectrics with KiCad's default ϵ_r 4.5 / $\tan\delta$ 0.02 or no material	<code>stackup.dielectrics</code>

Gap	Priority	Category	What is missing	Where it goes
nets-without-target	high	I, C	nets that look like high-speed interfaces by their name, without a target impedance (confirm first)	net_classes.<class> (a class without a target), or a net class in KiCad
classes-without-standard	medium	C	classes with a target but only a generic profile (1 Gbit/s, generic limits)	net_classes.<class>.profile
assembly-variant	high	C, P	the board has assembly variants (or forks where all alternatives are fitted) and none is chosen	variant, variants
assembly-variant-forks	medium	C	the chosen variant still has forks where all alternative series parts are fitted	variants.<name>.not_fitted (or sheets, bom)
channel-part-values	medium	C	parts in signal channels whose value kisi cannot read ("DNP", a part number): modelled as open	the Value field in KiCad (not kisi.json)
power-tree	medium	P	no regulator chain (PI_REG): no budget, no regulator load check	fields: PI_REG
rails-without-voltage	medium	P	supply rails (found by name, pin name or a field) without a known voltage	NAME@<V>V in a PI_REG entry
rails-without-source	medium	P	supply rails without a source: no IR drop	PI_REG (input or regulator output), else PI_SOURCE
regulator-efficiency	low	P	switching regulators without an efficiency: 85 % assumed	PI_REG (buck 90%)
ic-loads	medium	P	ICs on supply rails without a load current	fields: PI_LOAD
cap-ratings	low	P	decoupling capacitors of 100 nF and more without rated voltage or dielectric	PI_CAP or the Value field (10u 6.3V X5R)
bead-ratings	low	P	ferrite beads on supply rails without an impedance rating	PI_SERIES or the Value field
cap-values	low	P	capacitors on supply rails whose value kisi cannot read: left out of Z(f)	the Value field in KiCad (not kisi.json)
thermal-data	low	P	parts with 0.1 W or more without thermal data: package defaults, Tj max 125 °C	PI_THERMAL

Categories: I impedance, C channel, P power (with the temperature estimate). nets-without-target and rails found only by name in rails-without-voltage can be dismissed.

Where to look, per kind of value:

- **Laminate:** the material name in the stackup → the manufacturer's datasheet, at a frequency near the signals (1–10 GHz) and for the resin content / glass style of the layer; better the fabricator's stackup report.
- **Interface profile and target:** the datasheets of the devices at the bus ends (interface type, data rate), the interface standard or the device's layout guide.
- **Loads (PI_LOAD):** device datasheets, supply current per rail in the expected operating mode; FPGAs and SoCs: the vendor's power estimator.
- **Regulators (PI_REG):** topology, efficiency at the expected load (often only a curve – read it there and say so), rated output current; input and output rails from the schematic. A PMIC gets one entry per channel.

- **Capacitors, beads, thermal data:** the part's datasheet – DC-bias curve, impedance at 100 MHz and DC resistance, the Thermal Information table ($R_{\theta JB}$, $R_{\theta JC}(\text{top})$, ψ_{JT} , $T_j \text{ max}$).
- **Assembly variants:** the project's BOM and position files per variant, the sheets that hold each option; never pick a variant silently.
- **Connector ends:** what the connector meets (termination or driver) and the channel behind it – loss and delay of a cable or mating board, or the vendor's Touchstone file.

The syntax of every field and kisi.json key: [Field and kisi.json reference](#).

Four ways to fill it in

- **In the viewer:** the "Prepare PCB" page (below).
- **By hand:** edit `<board>.kisi.json` with the skeleton from `config-template`, check with `config-check`.
- **In KiCad:** the `PI_*` fields can live on the schematic symbols instead ("Update PCB from schematic" copies them to the footprints). kisi.json wins where both exist; sources are only kept in kisi.json.
- **With an agent:** the `kisi-prepare skill` runs the gap list, researches the values with sources and writes the kisi.json; you review the result.

In the viewer: Prepare PCB

The screenshot shows the 'Prepare PCB' interface in the kisi viewer. The interface is divided into several panels. On the left, a 'PREPARE THE BOARD' panel shows a checklist for 'Impedance' (2 of 2 done), 'Channel' (1 of 4 done), and 'Power' (0 of 4 done). Below this is a 'CHECK' panel with 'ok - 2 entries'. The main area is titled 'THUNDERSCOPE_REV5.3.KISI.JSON' and contains a 'Ready for the impedance check' section with a green checkmark and a 'Nets without a target impedance' section with a green checkmark. A 'Laminate data' section is also present with a green checkmark. On the right, a code editor shows the JSON configuration for the board. At the bottom, there are buttons for 'FILE Laminate (stackup)' and 'FILE Net class targets'.

"Prepare PCB" in the top bar (between the example boards and "Load files") opens what the board file lacks – with its kisi.json, if one is loaded – as a **checklist** in three tabs: Impedance, Channel and Power (with the temperature). The badge on the button counts the open points.

Each gap is a step, ordered by priority, with a marker: while it is open, the number of its items that are still open (seven ICs without a current: 7), red or amber by priority (low-priority steps are optional refinements); a green check mark when it is done, an arrow when it is with the agent, a dash when it is dismissed. Steps keep their place when one is done. An open step says in one line what it needs ("Enter the current per rail for 7 ICs, with a source") and shows only those inputs; *more* holds where to find the values, where they go and optional fields. **Apply** runs the gap list through the core at once: the FILE gets its check mark when kisi no longer reports it (or reports only dismissed

items), and the next open step opens; otherwise the step stays open and says what is still missing ("6 of 7 ICs still missing: ..."), with the check's errors for its entries. "hand to the agent" settles a step for now and adds it to the agent card's prompt; "hand single items to the agent" hands over only some of its items (the step's marker counts the rest). The tab shows its progress ("1 of 2 required steps done") and ends with "Ready for the impedance check" (or the channel check, the power analysis) and a button to that view; the last step is saving the kisi.json. Where a gap mixes categories, a tab shows only the items that matter for it (a dielectric with only a default $\tan\delta$ is a Channel item). Nets in "Default" that look like an interface get a class of their own in the step (name, nets or patterns, target, profile – written as `net_classes.<name>.nets`); dismissing is for nets that are not interfaces. Below the checklist, in `<board>.kisi.json` holds everything the category uses, loaded and entered, as editable tables:

Tab	Forms
Impedance	laminates per dielectric (material, ϵ_r , $\tan\delta$, thickness) and the solder mask ϵ_r ; net class targets (Ω , differential)
Channel	the laminate; net class → interface profile; connector ends (model, R/C/L, the channel behind as loss and delay or a Touchstone file with its port order and ports per pin); the assembly variant to analyse and variant definitions (sheets, parts, BOM, BOM exceptions)
Power	<code>PI_REG</code> (with one-click suggestions for rails behind a ferrite or link), load currents per rail, <code>PI_SOURCE</code> , <code>PI_SERIES</code> (Z at 100 MHz, DC resistance), <code>PI_CAP</code> (rated voltage, dielectric, ESR, ESL – one row per part number or value and package, written on every capacitor of the group, with the parts' own Voltage Rating / Dielectric fields as a one-click suggestion), <code>PI_THERMAL</code> , <code>PI_TARGET</code> and any other field

Every entry has its own source (kind, document; url, page and note behind "..."); a table's shared source fills the rows entered without one. Name-based gaps can be dismissed with a reason; a dismissal that this board does not report (any more) is listed next to its gap, or under "Stale dismissals", with "remove". **Edits made in the views** – the stackup and the class targets in the Impedance view, profiles and connector ends in the Channel view, the power tree or rail setups in the Power view – are listed on their tab and move into the draft with one action and a source (the power settings as `PI_REG` / `PI_LOAD` / `PI_SOURCE`, through the same conversion as the core's field reader).

The draft is the board's kisi.json itself: the loaded file with your edits, one for all three tabs (the laminate entered under Impedance also closes the gap under Channel). It is kept per board in this browser until it is saved or discarded, so a reload of the page and of the board brings it back; the side card says how it differs from the loaded file, and "Discard draft" returns to it. A draft belongs to the kisi.json it was made on (by content): when that file has changed since – edited elsewhere, or a newer build of kisi – the draft is not applied, and a line above the tabs offers "Restore" (the draft over the loaded file) or "Discard". A draft made on a bundled example is dropped when the example's kisi.json changed. The check (the same as `kisi config-check`) runs as you type, the gap list follows a moment later, and the right column shows the resulting file. "Save kisi.json" writes the draft, or the loaded file with only the current category's changes (the rest stays in the draft); keys the page has no form for are kept, and a file loaded and saved unchanged stays the same. Errors in what is saved block it, warnings do not. In Chromium a save dialog opens next to the board (and "Reload" reads the file from then on); other browsers download it. The saved file is loaded at once. Research-heavy gaps go to the agent path: what is handed over – whole steps or single items – is listed in the agent card, and its prompt for the kisi-prepare skill names each gap with what is needed, where it goes in kisi.json and its items (refdes with part number and rails, nets, rails, classes, layers), limited to that category (`kisi gaps --for ...`) or for all categories. With nothing handed over the prompt asks for the category's open gaps. The marks are kept with the draft in this browser, not in kisi.json; a category whose open gaps are all handed over says so.

Worked example: Seeed Studio XIAO nRF54LM20A

`docs/examples/xiao-nrf54lm20a/XIAO nRF54LM20A.kisi.json` is a board configuration that describes the board's **power tree for the USB supply case**, every value with its source. The kisi-prepare skill wrote it from the datasheets, and it was then cleaned up by hand. It shows a PMIC with several channels, load switches, a 0 Ω link and ferrite beads as pass elements, hierarchical net names in quotes, and loads marked as typical, maximum or estimated.

The board

- **Seeed Studio XIAO nRF54LM20A**, hardware V1.0: the KiCad 10 project `XIAO_nRF54LM20A_V1.0_SCH&PCB_260508`, six layers. Seeed publishes one project for the XIAO nRF54LM20A and the Sense variant, so the board file includes the IMU and the microphone.
- The hardware files are by Seeed Studio, licensed CC BY-SA 4.0, and are **not included** in kisi. Get them from the Resources section of the [XIAO nRF54LM20A wiki page](#) ("XIAO nRF54LM20A KiCad Project", [direct link](#)).

Step 1: what the board file alone gives

Unpack the project and run the gap list without a configuration:

```
node kisi.mjs gaps "XIAO nRF54LM20A.kicad_pcb"
```

With kisi 0.3.0 it reports no high-priority gap, five medium and two low ones:

Gap	Items
stackup-defaults	all 5 dielectrics carry KiCad's defaults (FR4, ϵ_r 4.5, $\tan\delta$ 0.02)
power-tree	no <code>PI_REG</code> field: no budget, no regulator loading
rails-without-voltage	<code>/03 Power/LD01_IN</code> (found by a pin name of U2), <code>/06 Peripherals/IMU_VDD</code> and <code>/06 Peripherals/MIC_VDD</code> (behind FB3 / FB4 – the gap suggests the <code>pass</code> entry), <code>VBAT</code> , <code>VBUS</code> , <code>VSYS</code>
rails-without-source	<code>3V3_OUT</code> , <code>IMU&MIC_3V3</code> , <code>SAMD11_3V3</code> , <code>TYPEC_5V</code> , <code>VSYS_3V3</code>
ic-loads	11 ICs, among them U8 nRF54LM20A, U10 flash, U11 IMU, U12 microphone, U5 SAMD11 – and U4, whose Value field is the manufacturer's name; the gap names its MPN UM3204H
cap-ratings	35 capacitors of 100 nF and more (with their MPNs)
bead-ratings	FB3, FB4 (Murata BLM03AG121SN1D)

The board is a small Bluetooth module without impedance-controlled interfaces, so the work is the power tree.

Step 2: the power tree

Read from the schematic (sheet 03 Power) and the datasheets:

```
USB1 USB-C: TYPEC_5V, 5 V input
└ U6 TPS22916 load switch → VBUS
  └ U2 nPM1300: VSYS follows VBUS (5 V while USB is present)
    └ U2 BUCK2 3.3 V → VSYS_3V3: U8 nRF54LM20A, U10 flash, LED1, U4 level translator
      └ U3 TPS22916 load switch → SAMD11_3V3: U5 SAMD11 debugger, U4
        └ R14 0 Ω → /03 Power/LD01_IN
          └ U2 LD01 3.3 V → IMU&MIC_3V3
            └ FB3 → /06 Peripherals/IMU_VDD: U11 LSM6DS3TR-C
              └ FB4 → /06 Peripherals/MIC_VDD: U12 MSM261DGT006
            └ U1 TPS628438 3.3 V → 3V3_OUT (header pin only; no load modelled)
```

As `PI_REG` fields in the kisi.json (sources left out here):

Part	PI_REG	Why
USB1	<code>TYPEC_5V input</code>	the supply input (assumption: 5 V USB)
U6	<code>TYPEC_5V > VBUS@5V pass r=80m 2A</code>	load switch, 80 mΩ maximum on-resistance; gives VBUS its voltage

Part	PI_REG	Why
U2	<code>VBUS > VSYS@5V pass id=U2-VSYS; VSYS > VSYS_3V3 buck 200mA out=L2 id=U2-BUCK2; "/03 Power/LD01_IN" > IMU&MIC_3V3 ldo iq=12.5u 50mA id=U2-LD01</code>	the PMIC: three channels separated by ; , each with its own id= ; BUCK2's output current leaves through L2
R14	<code>VSYS > "/03 Power/LD01_IN" pass</code>	0 Ω link, a hierarchical name in quotes
U1	<code>VSYS > 3V3_OUT buck 600mA out=L1</code>	no efficiency in the datasheet as a number: kisi's 85 % applies (estimate)
U3	<code>VSYS_3V3 > SAMD11_3V3 pass r=90m</code>	load switch for the debugger
FB3, FB4	<code>IMU&MIC_3V3 > "/06 Peripherals/IMU_VDD" pass r=0.5 (and MIC_VDD)</code>	ferrite beads, 0.5 Ω DC resistance (maximum)

Then the loads (PI_LOAD) and the beads' PDN data (PI_SERIES z100=120 rdc=0.5):

Kind	Values
Datasheet	U8 12 mA (radio TX at maximum power plus CPU, typical); U4 12 μ A + 2.4 μ A; U11 0.9 mA (typical); FB3/FB4 120 Ω at 100 MHz and 0.5 Ω DC resistance (maximum); on-resistances of U6 and U3, the nPM1300 channel ratings and LD01's 12.5 μ A ground current, VSYS following VBUS (Nordic DevZone answers by Nordic staff); TPS628438 3.3 V setting and 600 mA
Estimate	U10 6 mA (from the datasheet of a sibling flash, the P25Q64H); U5 4.4 mA (from the SAM D20 table, the same core); U12 0.75 mA (from sibling microphones); LED1 1.2 mA (from its series resistors); the efficiency of BUCK2 and U1 (kisi's default 85 %)
Assumption	USB1 supplies 5 V; the R14 0 Ω link counts as 0 Ω

The **note** of each source says whether a value is typical or maximum and at which condition.

Step 3: check and run

Copy the kisi.json next to XIA0 nRF54LM20A.kicad_pcb (same stem):

```
node kisi.mjs config-check "XIA0 nRF54LM20A.kicad_pcb" # ok, 17 entries
node kisi.mjs power "XIA0 nRF54LM20A.kicad_pcb" # power tree, IR drop, losses
node kisi.mjs gaps "XIA0 nRF54LM20A.kicad_pcb" # what is still open
```

In the viewer: drop the .kicad_pcb, the .kicad_pro and the .kisi.json together into kisi.html, then open the Power view.

With kisi 0.3.0, kisi power gives 100 mW from USB: 83 mW to the loads and 17 mW of losses. BUCK2 loses 14 mW at the estimated 85 % and LD01 3 mW. The largest DC drop is 0.17 mV, on VSYS_3V3.

What is still open

kisi gaps then lists these; they are expected for this example:

- rails-without-voltage: VBAT.** The battery is not part of the USB-case tree. On battery, VSYS is VBAT (3.0–4.2 V) and the SAMD11 debugger is off; kisi has one tree per board for now (ROADMAP: power tree per supply case). The nPM1300 battery charger is not modelled either – while it charges, its current is a load on VSYS.
- ic-loads: U7,** the XIA0 header. External loads on the 3V3 and VBUS pins are unknown.
- regulator-efficiency: BUCK2 and U1.** The datasheets give efficiency only as curves.
- cap-ratings:** the board's capacitors carry Voltage Rating and Dielectric fields, but kisi does not read them yet (ROADMAP: capacitor ratings from part fields), so they are not repeated as PI_CAP here. kisi gaps suggests them per capacitor (suggest), and the viewer's Prepare page writes them as PI_CAP for a group of parts at once.

- `stackup-defaults` : the laminate values are KiCad's defaults; for a board without impedance-controlled interfaces they matter little.

Field and kisi.json reference

Everything kisi reads besides the copper: names and fields in the KiCad project, the board configuration `<board>.kisi.json`, and the other files the CLI and the viewer accept. How to fill these in step by step is in [Preparing a board](#); the commands are in the [CLI reference](#).

What kisi reads from the KiCad project

- **Board file** `.kicad_pcb` (KiCad 8, 9 and 10): copper, zones, vias, footprints with their fields, the stackup, DNP and "exclude from BOM" attributes, the sheet of every part and – KiCad 10 – the assembly variants (variant list and per-part overrides: DNP, BOM, field values). kisi never writes into KiCad files.
- **Project file** `.kicad_pro` next to the board (same stem): the net classes and which nets belong to them. Without it, no net class has a target impedance unless `<board>.kisi.json` gives one.

Target impedance from net class names

A class name with a number followed by `Z`, `R`, `Ohm` or `Ω` (or `Z` followed by a number) between 20 and 200 gives the target: `50Z`, `FE_50Z_INPUT`, `USB_90R`, `Z85_PCIe`. The target is differential when the name contains `Diff` or `DP` (`90Z_Diff`, `FE_100Z_Diff`), or when it is 80 `Ω` or more and the name contains `USB`, `LVDS` or `PCI`. Without a number, interface keywords give an assumed target, marked "interface assumption":

Keyword in the class name	Target
USB	90 Ω differential
PCIE, PCI_E	85 Ω differential
LVDS, HDMI, TMDS, SATA, MDI, ETH	100 Ω differential
RF	50 Ω single-ended

`net_classes` in `<board>.kisi.json` gives or overrides the target of an existing class. With `nets` it also gives nets to a class – names or patterns with `*` and `?`, matched against the full net name and its last part, like KiCad's net class patterns – and defines the class if the project has none of that name: nets that sit in "Default" get a target without a change in KiCad, e.g. `"QSPI": {"nets": ["*QSPI_*"], "target_ohm": 50, "diff": false, "profile": "se50", "source": {...}}`. `config-check` reports a pattern that matches no net and a net given to two classes.

Interface profile

The channel check judges every lane against an interface profile (transmitter, receiver, limits) from `data/interfaces.json`. kisi assigns one automatically – by net name first, then by class name, the first matching rule of the file – unless `net_classes.<class>.profile` in `<board>.kisi.json`, the viewer's choice or `--class CLASS=PROFILE` sets one. `kisi profiles` lists the profiles; every value in the file carries its source (standard, vendor guideline, or a kisi guide value).

Part numbers

Where the Value field does not show the part number (`100n`, a manufacturer name), kisi takes the first non-empty MPN field – `MPN`, `Manufacturer Part Number`, `MFR PN`, `Mfg Part Number`, ... (case, spaces and punctuation ignored) – and shows it in `kisi gaps`, `kisi interfaces` and the power tree, so that the datasheet can be found.

Supply rails

A net is a supply rail when its name carries a voltage (`+3V3`, `1V8`, `-5V`; not signal names such as `PG_1V0`), when a power field names it, or – for `kisi gaps` – when its name or a pin name on it looks like a supply (`VBUS`, `VSYS`, `VBAT`, `VCC...`, `VDD...`, `VIN...`, `VIO`, ...). A rail without a voltage in its name needs one: `NAME@<V>V` in a `PI_REG` entry (below).

Footprint fields

Fields on the symbol, copied to the footprint by "Update PCB from schematic" – or written into `<board>.kisi.json` (fields, below) when the KiCad files should stay untouched. Field names are not case-sensitive.

Field	Example	Meaning
PI_REG	+12V input	supply input (a connector): source of the power tree and of the IR drop
	+12V > +5V buck 90% 3A out=L3	switching regulator: efficiency, rated output current, the part that carries the output
	+5V > +3V3 ldo iq=2mA 1A	linear regulator with its quiescent (ground) current
	+3V3 > +3V3A pass r=0.05	ferrite, 0 Ω link or shunt with its series resistance (Ω)
	+5V > VCORE@0.9V buck 85% id=U4A	rail without a voltage in its name; one IC with several channels: entries separated by ; , each with its own id=
	+3V3 > "/Sensors/IMU VDD" pass r=0.5	net name with spaces (hierarchical name) in double quotes, also "/Power/VDD CORE"@0.9V
PI_LOAD	+1V0=2.5A, +1V8=300mA or +1V0=2W	load current (or power) per rail
	+5V=60mA@L32/L33	current that enters the board at other parts' pads (e.g. the inductors of an output stage); the IC still returns it into ground
	"/Sensors/IMU VDD"=0.9mA	net name with spaces in double quotes
PI_SOURCE	+3V3 or +3V3=3.3V	source pads for the IR drop when there is no regulator chain (several separated by , or ;)
PI_CAP	esr=5m esl=0.45n rated=6.3V X5R c@3.3V=2.1u	decoupling capacitor data for the PDN from its datasheet; replaces the package estimates and the DC-bias curve
PI_SERIES	z100=600 rdc=0.15 · l=2.2u dcr=50m · r=10m	ferrite bead, inductor or resistor in a supply path, for the PDN per IC (replaces the typical values)
PI_TARGET	VCCINT=10m fmax=50M or 15m0hm	target impedance of an IC for a rail and the band it applies to (default: from the IC's own current step)
PI_THERMAL	theta_jb=17.4 theta_jc=53.6 psi_jt=1.1 tj_max=125	thermal data from the datasheet's Thermal Information table (K/W, °C) for the temperature estimate

PI_REG in detail

One entry per regulator channel: `INPUT > OUTPUT kind [values]`, or `OUTPUT input` for a supply input. `>` may also be written `->` or `→`.

- **kind:** `input`, `buck`, `boost`, `buck-boost`, `sepic`, `inverter`, `ldo`, `pass`. Without a kind, an entry with an input rail is a `buck`, one without is an `input`.
- **values:** efficiency `90%` (or `eff=0.9`) – without one, a switching regulator gets kisi's default of 85 %, marked as an estimate everywhere; rated output current `3A`, `500mA` (or `imax=`); `iq=1mA` quiescent current drawn from the input; `r=10m` series resistance of a pass element (Ω); `out=L3` / `in=...` the part whose pads carry the output / input current (an inductor or diode), if not the regulator itself; `id=U4A` a unique id when one part has several channels; `part=...` a part name for the lists.
- **voltages:** from the net name (`+3V3`, `1V8`, `-5V`), else `NAME@<V>V` wherever the rail appears.

PI_CAP, PI_SERIES, PI_TARGET, PI_THERMAL keys

Field	Keys
PI_CAP	<code>c</code> = capacitance, <code>esr</code> =, <code>esl</code> =, <code>rated</code> = (also <code>v</code> =, <code>vr</code> =) rated voltage, <code>c@<V></code> = capacitance at a DC bias (several allowed), a dielectric word(<code>X5R</code> , <code>X7R</code> , <code>C0G</code> , ...)
PI_SERIES	<code>r</code> = (also <code>dcr</code> =, <code>rdc</code> =) DC resistance, <code>l</code> = inductance, <code>z100</code> = bead impedance at 100 MHz, or the parallel model <code>rp</code> =, <code>lp</code> =, <code>cp</code> =
PI_TARGET	<code>RAIL=<Q></code> per rail (several separated by <code>;</code> or <code>,</code>), or a bare value for any rail; <code>fmax</code> = upper end of the band
PI_THERMAL	<code>theta_jb</code> =, <code>theta_jc</code> = (JEDEC two-resistor model), <code>psi_jb</code> = when there is no <code>theta_jb</code> , <code>psi_jt</code> = for the case top, <code>theta_ja</code> = only as a fallback (it describes the JEDEC test board), <code>tj_max</code> =, <code>p=0.2W</code> the heat that stays in the part (replaces the power tree's value, e.g. for a connector that passes power on)

Values take SI prefixes (`5m`, `0.45n`, `2.2u`, `50M`); resistances also `4R7`, `2k2`, `10mR`.

Net names in fields

Net names are written as in KiCad, or as the last part of a hierarchical name (`IMU_VDD` for `/Sensors/IMU_VDD`) when that is unique on the board. A name with spaces or one of `;`, `,`, `=`, `@`, `>` goes in double quotes (`PI_REG`, `PI_LOAD`, `PI_SOURCE`, `PI_TARGET`); inside `<board>.kisi.json` the quotes are escaped as `\`.

Board configuration `<board>.kisi.json`

Values the board file lacks or holds only as KiCad defaults, each with its source. The file sits next to the board with the same stem (`board.kicad_pcb` → `board.kisi.json`); CLI and viewer lay it over the board when they load it. `kisi config-template <board>` writes a skeleton with this board's dielectrics, net classes and assembly variants; `kisi config-check <board>` checks it (the viewer runs the same check when the file is dropped).

```
{
  "kisi_config": 1,
  "board": "board.kicad_pcb",
  "stackup": {
    "dielectrics": [{"index": 0, "epsilon_r": 3.65, "loss_tangent": 0.0095,
      "source": {"kind": "datasheet", "document": "Isola FR408HR", "page": "2", "note": "10
GHz, typical"}}],
    "solder_mask": {"epsilon_r": 3.8, "source": {"kind": "assumption", "note": "typical LPI mask"}}
  },
  "net_classes": {"PCIE": {"target_ohm": 85, "diff": true, "profile": "pcie3", "source": {"kind":
"standard", "document": "PCIE CEM 3.0"}}},
  "connector_ends": {"P1": {"model": "profile", "rest_loss_db": 6, "source": {"kind": "estimate", "note":
"1 m cable"}}},
  "fields": [{"ref": "U5", "name": "PI_LOAD", "value": "+1V0=2.5A", "source": {"kind": "datasheet", "url":
"...", "note": "maximum"}}],
  "variant": "TS-PCIE",
  "variants": {
    "TS-PCIE": {"description": "PCIE card", "not_fitted_sheets": ["/TS-USB4 Components/"], "fitted":
["C101", "C102", "C128"],
      "source": {"kind": "design", "document": "TS-PCIE_MP_BOM.csv"}},
    "TS-USB4": {"bom": "TS-USB4_MP_BOM.csv", "bom_exceptions": ["TP*"], "source": {"kind": "design",
"document": "TS-USB4_MP_BOM.csv"}}
  },
  "notes": ["what was assumed and why"]
}
```

Key	Content
kisi_config	format version, must be 1
board	the board file it was written for; a different board is an error in config-check
stackup	per dielectric (index 0 = the layer below F.Cu): material, epsilon_r (1...20), loss_tangent (0 ... 0.2), thickness (mm); solder_mask.epsilon_r
net_classes	per net class: target_ohm (10 ... 200; differential when diff is true), diff, profile (an id from kisi profiles), nets (names or patterns given to this class; defines a class the project does not have)
connector_ends	per connector where lanes leave the board: what sits there and the channel behind it (below)
fields	footprint fields laid over the board: ref, name (PI_*), value
variant	the assembly variant to analyse: a KiCad 10 variant of the board, one of variants, or default (the board file as it is)
variants	assembly variants for boards without KiCad 10 variants, or on top of one of the same name (below)
gaps	dismissed: items of name-based gaps that do not apply, with a reason (below)
notes	free text: what was assumed and why

Every value needs a source: kind plus at least one of document, url, note (page optional). Kinds: datasheet, standard, application_note, measurement, simulation, design (the project's own BOM, position or schematic files), estimate, assumption. Values of kind estimate or assumption are reported as such. Say in the note whether a value is typical or maximum and at which condition.

Precedence: board file < <board>.kisi.json < edits in the viewer / --class, --variant and --stackup on the command line.

connector_ends

A lane from a device to a connector is judged on the board share – loss, return loss, skew (a lane from connector to connector as well, without an eye). Its eye needs to know what sits at the connector:

```
"connector_ends": {
  "P1": {"model": "profile" | "matched" | "high_impedance" | "custom",
    "r_ohm": 50, "c_pf": 1.5, "l_nh": 0.5,
    "rest_loss_db": 6, "rest_delay_ps": 500,
    "touchstone": {"file": "cable.s4p", "port_order": "1-3,2-4", "flip": false},
    "source": {"kind": "estimate", "note": "..."}}
}
```

- **model**: the termination when the connector receives, the driver when it sends – the profile's (profile, default), matched, high_impedance (with c_pf), or custom (r_ohm per line, 0 = high impedance, with c_pf, l_nh).
- The channel behind the connector: a lossy line with rest_loss_db (loss at the profile's Nyquist frequency, positive dB) and rest_delay_ps (default 500), or a Touchstone file. With either, the eye covers the whole channel and is judged; without, the eye covers the board share only and is information.
- **touchstone**: file (path relative to the kisi.json; in the viewer, drop the file with the board); .s2p for single-ended lanes or on each line of a pair, .s4p for a pair with its mode conversion (port_order "1-3,2-4" – P1→3, N2→4, the default – or "1-2,3-4"); port 1 at the connector unless flip.
- One file for all lanes of the connector (.sNp): "ports": {"A16": [1, 9], "A17": [2, 10], ...} gives per connector pin the file's port at the connector and at the far end. The other lanes in the file are aggressors in each lane's eye – worst case, entering at the connector without their board traces (conservative); their directions are the guessed lane directions.

In the viewer the same settings are in the interface's card of the channel overview.

variants

Which parts are fitted, per variant. Rules apply in this order, later ones win:

1. the board file (DNP, and the KiCad 10 variant of the same name),
2. `bom`: a CSV next to the `kisi.json` with a "Reference" or "Designator" column – parts not in it are not fitted, except parts marked "exclude from BOM" in KiCad and `bom_exceptions` (gold fingers, test points),
3. `fitted_sheets` / `not_fitted_sheets`: a sheet covers its sub-sheets, the most specific one wins,
4. `fitted` / `not_fitted`: references, `*` and `?` allowed.

A part that is not fitted connects nothing; its pads stay as copper and become stubs on the lanes, and nets that only connect parts not fitted are not checked. When a board has variants – or forks where all alternative series parts are fitted – and none is chosen, the lane commands stop with the list (CLI) or ask (viewer); see [Getting started](#). In the viewer, drop the BOMs together with the board and the `kisi.json`.

gaps.dismissed

Some gaps are guesses from names: nets that look like an interface by their name (`nets-without-target`) and rails found only by their name (`rails-without-voltage`, items with `"dismissable": true`). Items that are not what the name suggests can be dismissed, with a reason:

```
"gaps": {"dismissed": [
  {"gap": "nets-without-target", "items": ["/FPGA/QSPI_DQ0", "/FPGA/QSPI_DQ1"],
   "reason": "QSPI configuration flash at 50 MHz, short lines - no impedance control needed",
   "source": {"kind": "design", "note": "optional"}}
]}
```

`kisi gaps` leaves dismissed items out (a gap whose items are all dismissed is gone) and lists them under `dismissed` with the reason, so a review still sees them. `config-check` requires one of the name-based gaps, a reason and nets that exist; an item the gap reports but not as a name-based guess (a rail found by a pin name) cannot be dismissed, and an item the gap no longer reports is a warning. Other gaps are filled or fixed, not dismissed.

Other input files

File	Used by	Format
<code>--stackup</code> FILE	all commands	stackup changes as the viewer stores them: <code>{"dielectrics": [{"index", "epsilon_r", "loss_tangent", "thickness"}], "mask_epsilon_r"}</code> – the same as <code>stackup</code> in <code>kisi.json</code> , without sources
<code>--tree</code> FILE	<code>power</code> , <code>thermal</code>	a power tree as JSON instead of the <code>PI_REG</code> / <code>PI_LOAD</code> fields: <code>{"rails": [{"net", "voltage"}], "elements": [{"refdes", "id", "kind", "from", "to", "eff", "iq", "r", "imax", "in_ref", "out_ref"}], "loads": [{"refdes", "net", "current", "basis", "note", "at"}]}</code> ; <code>tools/tree_to_fields.py</code> turns it into <code>kisi.json</code> fields
BOM(CSV)	<code>variants.*.bom</code>	a "Reference", "Designator" or "Ref" column (references separated by commas or spaces)
Touchstone (<code>.s2p</code> , <code>.s4p</code> , <code>.sNp</code>)	<code>connector_ends.*.touchstone</code>	Touchstone 1.x and 2.x

Output files

- `kisi report` writes everything into `analysis/kisi.json` next to the board (`--out` elsewhere), for file-based review tools and later sessions.
- `kisi net ... --s2p FILE / --s4p FILE` write the channel's S-parameters as Touchstone.
- `kisi export` writes a lane or a bus into `analysis/export/<name>` next to the board (`--out` elsewhere): the subcircuit library, test benches and S-parameter decks for ngspice, Xyce and LTspice, Touchstone files, a KiCad project and a README; the viewer's "Export..." gives the same files as a zip, as its export settings choose ([CLI reference](#)).
- The viewer exports Touchstone files per net and writes `<board>.kisi.json` from its Prepare PCB page ([Preparing a board](#)); its other edits stay in the browser.

Models and limits

What kisi computes, how, how well it agrees with references, and which numbers are estimates. kisi is a quick design check: its results are estimates without any warranty of correctness, not a replacement for sign-off simulation or measurement.

Scope: what kisi computes and what it leaves to other tools

kisi is a fast linear checker. It computes the board's lines, vias and channels as linear networks in the frequency domain and judges them against interface profiles with a linear transmitter and receiver: impedance, loss, return loss, skew, crosstalk estimates and a linear eye for every lane of a board in seconds, plus the DC and PDN analyses of the power side.

Out of scope, on purpose: IBIS and IBIS-AMI buffer models, equalisers (CTLE, FFE, DFE) and transient simulations with non-linear drivers (overshoot and ringback with real buffers, DDR timing flows, long patterns). For sign-off, export the channel ([Channel export](#)) – as Touchstone files (a lane, or a whole bus as one N-port) or as SPICE subcircuits – and simulate it with the buffers in a tool made for that: PyBERT, ngspice, Xyce or LTspice with IBIS models, KiCad's simulator.

Why kisi is small and fast

- **Linear networks in the frequency domain.** A line is a chain of 2×2 ABCD matrices per frequency (RLGC per cross-section), a via a small ladder, a part a few impedances; a lane is the product of these matrices – a few hundred multiplications per frequency point. Where a lane branches (a pair as a 4-port, receivers on a multi-drop line, an N-port file behind a connector) kisi solves one small linear system per frequency (nodal analysis with a handful of nodes). There is no Newton-Raphson iteration and no time stepping: the eye comes from the step response of the linear network.
- **Compared with a circuit simulator:** a lane takes kisi about 0.15 s (ThunderScope: 71 lanes in about 11 s on one thread, a few seconds on several cores). The same ThunderScope LVDS lane exported as a SPICE ladder has about 42 000 elements and takes 6–15 minutes in ngspice or LTspice. A reduced export (vector fitting or lossy-line elements) is planned to shorten that ([ROADMAP](#), block 3).
- **2D, not 3D.** Impedances come from formulas (a board in under a second) or from a quasi-static 2D field solver per cross-section – only for nets with a target, cached on disk and in the browser; vias, pads and planes use reduced models checked against 3D references instead of a full-board 3D solution.
- **Sparse and parallel where it counts.** The DC voltage drop, temperature and PDN grids use sparse solvers (`kisi-sparse` on faer); the channel check computes its lanes on several threads (native) or Web Workers (viewer) with results identical to one thread.
- **One engine, compiled once.** The models are Rust compiled to WebAssembly: the whole viewer including the field solver is one HTML file of about 3 MB that runs offline in a browser, and the same engine runs the Node CLI `kisi.mjs`.

Impedance

- **Formulas:** Hammerstad-Jensen, Wheeler, Wadell (CPWG), IPC-2141 – in a second for every track. Microstrip within 1–2 %; coupled and coplanar lines 5–15 % off (asymmetric stripline +11 %). A formula value over a cut-out in a layer between trace and reference is flagged "rough" and can be 25 % off.
- **2D field solver:** `js_2d_fields` by Henrik Forstén (GPL-3.0), quasi-static, included unmodified (see [UPSTREAM.md](#)); its linear solver runs in Rust (`crates/kisi-sparse`). Cross-sections are taken along every line with its reference planes (also across gaps), coplanar ground and the differential partner. Within 1.7 % of the HFSS/Ansys references in Forstén's test suite, ϵ_{eff} within 1 %.

- **Pads and gold fingers:** a track inside an SMD pad of its own net (a track runs to the pad's centre; a card-edge finger) has the pad's cross-section – its width and what lies below it, a cut-out in the plane included –, exactly from the pad's edge; the rest of a part's pad is a shunt capacitance from that cross-section (without one, an estimate per package from `data/components.json`). Copper with no plane above or below it on any layer – a finger with the planes cut back – takes its return in the coplanar copper beside it and a pair's partner (coplanar waveguide and coplanar strips without a ground plane, an estimate); with nothing within 0.5 mm (or 3 widths) beside it the channel uses $1.5 \times$ the target impedance. Against 3D: [TESTS.md](#) section 15.

Deviation on ThunderScope cross-sections (reference: Forstén's microstrip/stripline solver; the last row HFSS):

Cross-section	Formula	Field solver
Microstrip 213 μm , with solder mask	+1.4 %	0.0 %
Diff-microstrip LVDS (127/130 μm)	+7.8 %	−0.3 %
Diff-microstrip PCIe (191/127 μm)	+5.6 %	−0.3 %
Stripline In3, spacings 1:8	+10.3 %	0.0 %
Microstrip over a 3 mm cut-out (HFSS)	+17 %	−0.5 %

Lines

RLGC per cross-section, passive and causal by construction (the series impedance and the shunt admittance per metre are positive-real functions of frequency):

- **Conductor:** the DC resistance turning into the skin effect (Wheeler's incremental inductance, or the field solve's resistance), $R_{\text{dc}} \cdot \sqrt{1 + jf/f_c}$ with its internal inductance, times the causal Huray roughness factor (Bracken, DesignCon 2012) – its loss part is Huray's, its inductance part what Kramers-Kronig requires. The roughness presets `smooth`, `hvlp`, `standard` are estimates.
- **Dielectric:** one Djordjevic-Sarkar model (a wideband Debye distribution fitted to ϵ_r and $\tan\delta$ at 1 GHz) for ϵ' and ϵ'' – in the lines, the via ladders and the coaxial via sections alike.
- **Microstrip dispersion:** a lossless Lorentz term that follows Getsinger's rising ϵ_{eff} below a resonance above the band. A passive line cannot follow a rise of ϵ_{eff} with frequency except below a resonance; a microstrip on a thick dielectric, whose Getsinger corner lies inside the band (at 50 Ω and 25 GHz from about 0.4 mm), gets the rise at the top of the band and less below it.

RLGC within 2 % of HFSS.

A channel is a network of N-port elements solved by nodal analysis (`kisi_si::rf::network`): the line's ABCD cascade, a differential pair as a 4-port (coupled sections from the odd and even mode, via pairs, stubs on both lines paired up), receivers on side stubs, a Touchstone file behind a connector as an S-parameter block. For a single line it equals the cascade to numerical precision. TDR comes from S11 (Sdd11 for pairs).

Vias

A physics-based ladder per via (`kisi_si::via_pbm`): capacitances from an axisymmetric Laplace solve, the plane pairs as radial lines with the nearby return posts (vias and plated pins of ground and supply planes, the nearest 12 within 3 mm), pads and annular rings, antipads from the zone fills (non-round ones as an equivalent circle), pours on the via's signal layers as coplanar returns, stubs and back-drill, the junction to the traces; pairs with their partner. Passive by construction: the plane-pair matrices treat every barrel as a cylinder of uniform current, and a pair's modes are taken as $(Z_{\text{PP}} + Z_{\text{NN}})/2 \mp Z_{\text{PN}}$ (the earlier $Z_{\text{PP}} \mp Z_{\text{PN}}$ gave via pairs with returns on one side a negative resistance from a few GHz). Vias that do not cross a plane are coaxial sections. Compared with 3D results of kisi's own FDTD solver: via impedance in the TDR median 2.3 %, 90th percentile 5.5 % (single vias); pairs within 3.5° phase; stub notch within 3 %; the four board vias with a 3D reference within 0.03–0.44 dB up to 12 GHz. An estimate, not a 3D result: above 12 GHz the ladder follows the 3D results only where its return posts close the via's cage. A via

whose path crosses a ground/supply plane pair that no return via bridges gets a warning. `--via-cavity` (finite plane pairs) and `--via-returns` (return groups per plane pair) are experimental options.

Lanes and the channel check

- **Lanes:** `kisi_si::topology` follows nets through series resistors and AC-coupling capacitors from device to device (or to a connector). Series parts and terminations carry package parasitics from `data/components.json` (estimates). The transmit direction is a guess (the termination sits next to the receiver) and can be swapped.
- **Stubs:** copper of a lane net that leaves the path is an open side stub – to unused pads and test points, to the pads of parts that are not fitted (with their pad capacitance, an estimate), and – when all alternatives of a fork are fitted – through the other part to the other device. A stub hangs on the side of a via that matches its layer.
- **Multi-drop:** on a line along one path the devices on side stubs are receivers with the profile's receiver model; the worst eye counts. Multi-drop trees through series parts are listed, not checked.
- **Eye:** PRBS7 NRZ with the profile's linear transmitter and receiver, from the lane's network (pairs: differential drive of the 4-port; ends per line to AC ground, so the common mode sees the same termination). No equalisation, jitter or noise. The step response has a DC point (the resistances of the signal band; where the channel changes below the lowest frequency – a small coupling capacitor, a choke – that frequency stands for DC) and the Gaussian edge as window. Warnings when the band ends inside the edge spectrum (above 35 % left: the edge rings by about 2 % and more), when there is response before the signal can arrive (non-causal data, time aliasing) or the step has not settled, and when S-parameters are not passive.
- **At a connector:** the eye uses the end set in `connector_ends` and the channel behind it; without a channel behind the connector it covers the board share only and is information, not judged. Lanes from connector to connector have no eye (transmitter and receiver are off the board).
- **Verdict:** one function (`kisi_si::profile::judge`) for viewer and CLI. Limits marked "kisi" in the profile sources are guide values, not standards. Mode conversion (Scd21) is information, not judged.

Crosstalk

A coupling scan of every net with a target: NEXT and FEXT from the neighbouring tracks, critical from 1 % of the aggressor swing (pairs as differential noise, with the common-mode noise beside it). The estimate uses formula coupling; the field solver refines the critical couplings. Broadside overlaps between layers are detected and counted, not computed. Several aggressors at once are only combined for the lanes of one N-port file behind a connector.

Power

- **DC IR drop:** copper rasterised per layer, vias as barrels (25 µm plating); per rail and for the ground return, which is solved like a rail with the supply input as 0 V. Every part returns what it draws from the supply rails minus what it feeds into them (a buck converter takes current from ground).
- **Power tree and losses:** from `PI_REG/PI_LOAD`. The power a load draws counts as heat in the part (an estimate), regulator losses on the regulator, copper loss is I^2R from the DC solutions. A switching regulator without an efficiency gets 85 % (an estimate, marked).
- **PDN:** a lumped model of regulator, plane and capacitors with a plane model above 1 MHz; per IC the rail's copper on all layers as an AC grid with plane capacitance, capacitors with ESR/ESL and DC-bias derating (estimated per package unless `PI_CAP` gives the data), ferrites and series parts up to the regulators (typical values unless `PI_SERIES` gives them), the loop inductance of each capacitor to the IC, anti-resonances, and a target from the IC's current step (or `PI_TARGET`).

Temperature (estimate)

Steady state of the board in air. One sheet per copper layer on a grid over the board outline (copper fill of every net, laminate in plane 0.8 and through the thickness 0.3 W/m·K, open via barrels), natural or forced convection (flat-plate correlations) and radiation from both sides; heat from the copper loss of every cell and the parts. A part is a junction

node with θ_{JB} to the board under its pads and θ_{JC} to its top (JEDEC two-resistor model), with values from `PI_THERMAL` or package defaults by family and size (`data/components.json`, estimates); the case top from ψ_{JT} or that model. Per via the heat through its barrel; per hot part the split of its heat into vias, laminate and the pad layer. The IR drop at temperature solves the rails again with $\rho(T) = \rho_{20}(1 + 0.393\% / K \cdot (T - 20\text{ }^{\circ}\text{C}))$.

Not included: heat sinks, conduction into mounting holes, connectors or card guides, the enclosure (a device in a closed case runs hotter inside: emulate it with the case temperature as `--ambient` and the inner coupling as `--h`), the split of a regulator's loss between IC and inductor, transients. Junction temperatures with package defaults can be off by tens of percent of the rise. Concept, physics and validation: [research/thermal.md](#).

Channel export

`kisi export` and "Export..." in the viewer's lane detail and bus card write a lane's board channel – as the check models it – for circuit and RF simulators:

- **SPICE subcircuit:** line sections as RLGC ladders (20 sections per wavelength at the top frequency) with Foster/Debye networks fitted to $R(f)$, $L(f)$, $G(f)$, $C(f)$ – positive elements only, so passive and causal; vias as kisi's via ladder; parts with their estimated parasitics (e.g. an AC-coupling capacitor as R–L–C with its pads); open stubs; a pair with both lines and its coupled sections as even/odd chains between ideal mode converters.
- **Test benches** for ngspice, Xyce and LTspice (PRBS7 between the profile's linear transmitter and receiver, the LTspice one with `.meas` results) and S-parameter decks; the netlist is shared, only the decks differ (the simulators call analyses differently; LTspice adds a series resistance (1 m Ω) and an implicit parallel damping to inductors by default, so the LTspice library sets `Rser=0` `Rpar=1e30` on every inductor to match ngspice and Xyce).
- **Touchstone** per lane (`.s2p`, `.s4p`) and a bus as one N-port with its port map.
- **Frequency range:** the ladders have 20 sections per wavelength at the top frequency (default 25 GHz; `--fmax`, or $5 \times$ Nyquist of the profile), which also ends the Touchstone grid. A lower top frequency shortens transients only so far: an 85 mm LVDS pair has 42 k elements at 25 GHz and 23 k at 2.5–5 GHz (ngspice 350 s and 180 s for two PRBS7 periods), because every section, via and part keeps its share.
- **KiCad project** (KiCad 7 format, opens in KiCad 7 and later; checked with KiCad 9's ERC): one symbol per element, parts with their value as a parameter (change `220n` in KiCad and the parasitics stay), or the lane as one symbol.

The terminations next to the receiver act as the receiver's termination and are not in the subcircuit. Not included: the channel behind a connector, crosstalk between lanes, IBIS ends. kisi's own model is passive and causal (checked over every lane of the example boards up to 25 GHz), so the fitted circuit can follow it across the band. Report: [docs/research/channel-export.md](#).

Accuracy levels

Every impedance and via result says which level it used and how far it typically is from kisi's validation (`data/accuracy.json`, with the sources; design: [research/accuracy-levels.md](#)):

Level	Typical deviation
2D field solver	impedance 2 %
formula: single microstrip / coupled or coplanar / stripline / over a cut-out	2 / 8 / 11 / 25 %
no plane (coplanar estimate)	15 %
via ladder: single vias / pairs (TDR, median and 90th percentile)	2.5 and 5.9 % / 12.6 and 14.4 %
coaxial via fallback	8.0 and 25.5 %

A net or lane takes the length-weighted mean of its cross-sections; the level is "field solver", "formula" or "mixed". The CLI writes it as `accuracy` in every `check` row and in `net`, the viewer as a muted " $\pm x\%$ " next to the impedance.

Verdicts are judged on the values; the uncertainty is shown beside them. Loss, return loss and the eye do not carry a number yet.

What is an estimate

kisi marks these as estimates in the viewer and in the CLI output:

- laminate $\epsilon_r/\tan\delta$ and thicknesses that are KiCad defaults (assumed in kisi overview), until `<board>.kisi.json` gives them;
- target impedances from interface keywords in class names ("interface assumption");
- copper roughness presets; package parasitics of series parts and terminations; the pad capacitance per package where a pad has no cross-section of its own; the coplanar impedance of copper without a reference plane;
- the transmit direction of a lane;
- regulator efficiency 85 % where none is given; the heat of a load (all drawn power stays in the part);
- capacitor ESR/ESL and DC-bias derating by package, with an assumed rated voltage where neither the Value text nor `PI_CAP` gives one; ferrite and inductor values without `PI_SERIES`;
- thermal resistances by package family and size, T_j max 125 °C, where `PI_THERMAL` gives none;
- values in `<board>.kisi.json` whose source kind is `estimate` or `assumption`.

Validation

- **2D field solver and line models** against the HFSS/Ansys cases of Forstén's test suite: impedance within 1.7 %, RLGC within 2 % (`tools/validate_field.sh`, which also compares against the stored baseline `validation/field_baseline.json`). Validation data: `validation/*.json` (cross-sections), `*.ref.json` (references from Forstén's solver, computed in Node with `tools/ref_forsten.mjs` and the solver shim `tools/forsten_sparse_shim.js`); `kisi-validate <cases.json> [<refs.json>]` compares the formulas, and the field-solver chain (`kisi-validate --geoms → tools/solve_xs.mjs → kisi-validate --field`) is what `validate_field.sh` runs.
- **Passivity and causality**: unit tests (series impedance and shunt admittance positive-real in the whole right half-plane; the pulse response of a 200 mm lossy line before its front; the plane-pair reductions with asymmetric returns) and a test over every lane of the example boards (`crates/kisi-core/tests/causality.rs`, in CI): the board's S-parameters passive up to 25 GHz and their pulse responses causal (details and numbers: [TESTS.md](#) section 14).
- **Vias and planes** against 3D results of kisi's own FDTD solver (`crates/kisi-fdtd`, checked against openEMS and analytic cavities): 74 reference cases in `validation/via_ref`, the test log in [validation/fdtd/TESTS.md](#), and the reference page [kisi-references.html](#) (attached to the releases in which the cases changed) with every case and kisi's models against it. Useful for teaching as well: it shows why vias and planes behave as they do.
- **Temperature solver** against analytic cases (1D slab exact, fin within 0.2 %, radial spreading in a plane with convection within 0.13 %, energy balance) and, with the datasheets' θ_{JB}/θ_{JC} on a model of the JEDEC JESD51-7 board, against their θ_{JA} (QFN-16 +4 %, SOIC-8 -15 %, SOT-23-5 -20 %).
- **Channel export**: ngspice reproduces kisi's own evaluation of the exported circuit (ThunderScope's LVDS lanes and nine PCIe lanes, to 0.001) and kisi's S-parameters within 0.03 dB (LVDS) and 0.10–0.34 dB (PCIe, strongly reflecting) up to $1.5 \times$ Nyquist; step responses within 2 % (`validation/channel_export/compare.py`, also in CI).
- **Still open**: validation against other solvers and against measurements. A measured test board is planned ([validation/test_board.md](#)). Until then, treat the numbers as plausible estimates, not as verified results.

Limits

- The channel check is a first look: linear transmitter and receiver from the interface profile, no jitter or noise, no simulation of several aggressors at once (except the lanes of one N-port file behind a connector). IBIS, IBIS-AMI

and equalisers are out of kisi's scope ([Scope](#)); a sign-off simulation uses the exported channel in a tool made for that ([Channel export](#)).

- 2D cross-sections and reduced via and plane models instead of a full-board 3D solution; board edges, via-to-via coupling and resonances of the whole board are not included.
- Data the board file does not hold (laminates $\epsilon_r/\tan\delta$, load currents, capacitor data) defaults to estimates; `kisi gaps` lists what is missing and `<board>.kisi.json` fills it in with sources ([Preparing a board](#)).
- One power tree per board: boards that run from USB or a battery have one tree per supply case, which kisi cannot represent yet.

CLI reference

kisi has two command lines with the same commands, arguments and JSON results (both run `kisi_core::commands`):

- `kisi.mjs` – one file for Node.js ≥ 18 , attached to every [release](#). It contains the viewer's engine and the 2D field solver.
- `kisi` – a native build (`cargo install --path crates/kisi-cli`) with formula impedances only (no field solver); useful for CI and development.

```
curl -fsSL0 https://github.com/riehlies/kisi/releases/latest/download/kisi.mjs
node kisi.mjs help
```

`node kisi.mjs <command> <board.kicad_pcb> [options]` prints JSON on stdout. Every result states the models and assumptions it used; progress (field solver, check rows) goes to stderr. An error prints `{"error": "..."}` and exits with code 1. The board's `.kicad_pro` and `<board>.kisi.json` next to it are read automatically.

kisi help

The text below is the output of `kisi help`. A test (`cargo test -p kisi-core`) fails when it differs from the program; `KISI_BLESS_DOCS=1 cargo test -p kisi-core docs_cli` writes the current text here.

`kisi <command> <board.kicad_pcb> [options] - JSON on stdout`

overview	stackup, impedance per net class (share within $\pm 5\%$, worst nets)
interfaces	buses and device-to-device lanes with profiles <code>[--class CLASS=PROFILE]...</code>
net <NET>	one net: impedance along the path, loss, TDR, vias, crosstalk; differential pairs also as 4-port (mixed mode, skew) <code>[--rise-ps 35] [--roughness smooth hvlp standard] [--s2p FILE --s4p FILE (Touchstone)]</code>
check	channel check of all lanes against interface profiles <code>[--class CLASS=PROFILE]... [--nets A,B] [--all] [--max 30]</code>
crosstalk	coupling scan (NEXT/FEXT from neighbouring tracks) of all nets with a target, critical from 1%; <code>kisi.mjs</code> refines the critical couplings with the field solver <code>[--nets A,B] [--all (all rows, not only critical)] [--max 30] [--rise-ps 35]</code>
profiles [ID]	interface profiles (no board needed)
power	regulator chain, DC IR drop per rail and ground return, losses, power flow <code>[--tree tree.json] [--pdn RAIL]</code>
thermal	temperature estimate from the power analysis: board temperature per layer, hot spots, junction temperature per part (PI_THERMAL or package defaults), copper temperature and IR drop at temperature per rail <code>[--tree tree.json] [--ambient °C (25)] [--h W/m²K per side] [--airflow m/s] [--vertical]</code>
report	write analysis/kisi.json next to the board <code>[--out FILE]</code>
export	a lane or a whole bus for circuit and RF simulators: SPICE subcircuits (RLGC ladders, kisi's via model, parts, stubs; pairs coupled) with test benches for ngspice, Xyce and LTspice, Touchstone (.s2p/.s4p, a bus as one N-port with port map) and a KiCad project with the profile's linear ends <code>--lane NAME --bus NAME</code> <code>[--format lib,spice,xyce,ltspice,touchstone,kicad (default all; lib: the subcircuits alone, spice: with the ngspice decks)]</code> <code>[--kicad-style detailed compact (one symbol per element, default; or the lane as one symbol)] [--fmax GHZ profile (top frequency of the ladder and the</code>

```

    Touchstone grid; default as the check, 25 GHz; profile: 5 x Nyquist]]
    [--fmin GHZ (grid start and step; default fmax/500)]
    [--out DIR (default analysis/export/<name> next to the board)]
    [--class CLASS=PROFILE]
gaps      what the analyses miss, where to find it, how to write it into <board>.kisi.json;
          per category (impedance, channel, power) whether there is anything to check and
          whether it is ready                                [--for impedance|channel|power]
config-check  check <board>.kisi.json (format, ranges, references, sources, variants)
config-template a <board>.kisi.json skeleton with this board's dielectrics, net classes and
               assembly variants

common options: --pro FILE (default: board.kicad_pro), --config FILE (default: board.kisi.json
next to the board), --no-config, --stackup FILE (stackup changes, as in the viewer),
--variant NAME (assembly variant: a KiCad 10 variant of the board or one of the kisi.json
"variants"; "default" = the board file as it is). Boards with variants - or with forks where
all alternative parts are fitted - need one for interfaces, check and report (else they stop and
list the variants); "variant" in <board>.kisi.json chooses one for every run.
--via-cavity (net, check, report; experimental, off by default): the plane pairs of the via models
as finite cavities - board edges, stitching vias, capacitors between supply and ground planes -
instead of infinite plates (validation/fdtd/BOARD_RESONANCES.md); slower.
--via-returns (net, check, report; experimental, off by default): return groups per plane pair in
the via models - a return via shorts only the pairs whose two planes it joins, crosses supply
planes as a floating return, capacitors between supply and ground planes nearby close a pair
(validation/fdtd/BOARD_RETURNS.md).
field solver (kisi.mjs only; overview, net, check, crosstalk, report): used by default for the
cross-sections of nets with a target, results cached on disk; --formula (no field solver),
--field-time S (time budget for new solves, default 120), --field-all (all cross-sections, not
only targets), --field-xt (also the coupling cross-sections of critical crosstalk victims, NEXT or FEXT
from 1 %;
net: always all of the net's)

kisi - a side project by David Riehl, started as a visualisation tool for the course Advanced PCB Design
at TU Darmstadt. GPL-3.0-only, https://github.com/riehlies/kisi
A quick design check: results are estimates, without any warranty of correctness.

```

Commands

overview

Stackup (with `assumed` for values that are KiCad defaults) and, per net class with a target impedance, the share of routed length within $\pm 5\%$ and the worst nets. `config` says which kisi.json was used. Runs the field solver.

interfaces

Bus and device-to-device lanes: pins, series parts, terminations, branches, side stubs (`stubs`: unused pads), the guessed transmit direction and the interface profile of each class (`--class CLASS=PROFILE` to try another). Needs an assembly variant on boards that have them (see below).

net

One net (`kisi net <board> <NET>`): its path (start, end, branches), impedance segments along it (layer, length, Z), loss, TDR extremes, vias (antipad, stub resonance, return path) and crosstalk from its neighbours; differential pairs also as a 4-port (mixed mode, skew). `--rise-ps` sets the edge (default 35 ps), `--roughness`

`smooth|hvlp|standard` the copper roughness preset; `--s2p FILE` writes the channel's S-parameters, `--s4p FILE` the pair's single-ended 4-port as Touchstone. Runs the field solver for the net's cross-sections. `accuracy`: the level and typical uncertainty of the impedance and the via models (as in `check`).

check

Channel check of all lanes against their interface profiles: loss, return loss, skew, via stubs and the eye; nets with a target but no devices on their own. Only yellow and red rows are printed (`--all` for green too, `--nets A,B` to restrict, `--max N` rows, default 30). `--class CLASS=PROFILE` overrides a profile (better: put it into `kisi.json`). Rows are computed on several threads (below); the result is the same as on one thread. Needs an assembly variant on boards that have them. Runs the field solver. Every row has `accuracy`: `impedance` (level formula / field solver / mixed, `uncertainty_pct` – the typical deviation from kisi's validation, length-weighted over the lane's nets –, `field_share`) and `vias` (`model` ladder / ladder (pair) / coax / mixed, `tdr_error_pct` median and p90), from `data/accuracy.json` (`models`).

crosstalk

Coupling scan of all nets with a target: NEXT and FEXT from the neighbouring tracks, critical from 1 % of the aggressor swing, with the worst aggressor per victim (`--nets A,B`, `--all` for every row, `--max`, `--rise-ps`). `kisi.mjs` refines the critical couplings with the field solver; `source`: `formula` values are estimates.

power

Regulator chain and budget (from `PI_REG/PI_LOAD` fields, `kisi.json` or `--tree FILE`), DC IR drop per rail and in the ground return (`ground_return`: `max_offset_mv` adds to the drop of the parts' rails), `losses` (heat per part, copper loss per rail and layer) and `flow` (power from the inputs through the regulators to the loads). `--pdn RAIL` adds the impedance $Z(f)$ of one rail against its target. Regulators flagged `efficiency_estimate` use kisi's default of 85 %.

thermal

Steady-state temperature estimate from the losses of `power`: board temperature per layer and hot spots; per part (`parts[]`) the junction temperature against T_j max and the case top, with `data_source` (`PI_THERMAL` or package defaults); heat through the vias and under each hot part, with `parts_hints` where thermal vias look short; per rail the copper temperature and the IR drop at that temperature. `--ambient` (°C, default 25), `--airflow` (m/s), `--vertical` (a vertical board), `--h` (surface coefficient per side in W/m²K, replaces the convection correlations, e.g. for the inside of a closed case). See [Models and limits](#).

report

Everything into `analysis/kisi.json` next to the board (`--out FILE` elsewhere); prints a summary. For file-based review tools and later sessions. Needs an assembly variant on boards that have them. Runs the field solver.

export

A lane (`--lane NAME`, a net of the lane) or a whole bus (`--bus NAME`, as `kisi` interfaces names it, e.g. "P1 ↔ U18") for circuit and RF simulators: the board channel as the check models it, as SPICE subcircuits with test benches for ngspice, Xyce and LTspice, Touchstone files (a bus also as one N-port with its port map) and a KiCad project with the interface profile's linear ends. `--format` chooses the files (comma-separated, default all): `lib` the subcircuit library alone, `spice` with the ngspice bench and S-parameter deck, `xyce`, `ltspice` (bench, and for a single-ended lane a `.net` S-parameter deck), `touchstone`, `kicad`. `--kicad-style` `detailed` (default: one symbol per element – lines, coupled runs, vias, parts with their value as a parameter, pads, stubs) or `compact` (the lane as one symbol). `--fmax GHZ` sets the top frequency of the ladders (sections per wavelength) and of the Touchstone grid (default: the check's 25 GHz), `--fmax profile` uses $5 \times$ Nyquist of the lane's interface profile (at least $1.25 \times$ the bit rate either way; a bus takes the highest of its lanes, so its N-port has one grid); `--fmin GHZ` the grid's start and step (default $f_{max} / 500$, at most 10000 points). `--out DIR` the folder (default `analysis/export/<name>` next to the board). The JSON lists the files, the frequency grid, the bench ends, how closely the circuit follows kisi's own S-

parameters, the element counts and notes. Needs an assembly variant on boards that have them. See [Models and limits](#).

gaps

Missing or default data: each gap has `id`, `priority` (high, medium, low), `area`, `issue`, `affects`, `views` (the analysis categories it matters for: impedance, channel, power – the temperature estimate is part of power), `where_to_find`, `write` (where it goes in `<board>.kisi.json`, with an example) and `items` (layers, classes, parts with their MPN; `"dismissable": true` on items found only by a name; items of gaps that mix categories carry their own `views` – a dielectric with only a default $\tan\delta$ matters for the channel, not the impedance – and the gap's `views` are then those of its items; capacitors in `cap-ratings` carry their `package` and, where the part's own fields name them, a suggested PLCAP value). `views` in the result says per category whether there is anything to check (`applicable`, `why`) and its `state`: `ready` (no open high or medium gap), `open`, or `nothing to check`. `--for impedance|channel|power` keeps the gaps of one category. `dismissed` lists the items left out because `gaps.dismissed` in `kisi.json` dismisses them, with the reason (and `not_applied` for items that are not reported). See [Preparing a board](#). Fast, no field solver.

config-check

Checks `<board>.kisi.json` against the board: format, ranges, references, sources, field syntax kisi cannot read, variants and the files they name, dismissed gaps. Prints `ok`, `errors`, `warnings`, the number of entries, and `messages` – every error and warning with the categories (`views`) of the entry it is about.

config-template

A `<board>.kisi.json` skeleton with this board's dielectric indices, net classes and suggested assembly variants (from forks where all alternative parts are fitted). Keep only the entries you fill in.

profiles

The interface profiles(`kisi profiles`), or one in full(`kisi profiles pcie3`). No board needed.

help

The text above. `kisi.mjs` also accepts `--version`.

Options

Option	Commands	Meaning
<code>--pro FILE</code>	all with a board	the project file(default: <code><board>.kicad_pro</code>)
<code>--config FILE</code>	all with a board	the board configuration(default: <code><board>.kisi.json</code> next to the board)
<code>--no-config</code>	all with a board	ignore <code><board>.kisi.json</code>
<code>--stackup FILE</code>	all with a board	stackup changes as the viewer stores them(format)
<code>--variant NAME</code>	all with a board	assembly variant: a KiCad 10 variant, one of the <code>kisi.json</code> <code>variants</code> , or <code>default</code>
<code>--class CLASS=PROFILE</code>	<code>interfaces</code> , <code>check</code>	interface profile of a net class (repeatable)
<code>--nets A,B</code>	<code>check</code> , <code>crosstalk</code>	only these nets
<code>--all</code>	<code>check</code> , <code>crosstalk</code>	all rows, not only the yellow/red or critical ones
<code>--max N</code>	<code>check</code> , <code>crosstalk</code>	at most N rows (default 30)

Option	Commands	Meaning
<code>--rise-ps PS</code>	<code>net</code> , <code>check</code> , <code>crosstalk</code>	edge rise time(default 35 ps)
<code>--roughness R</code>	<code>net</code> , <code>check</code> , <code>crosstalk</code>	copper roughness preset: <code>smooth</code> , <code>hvlp</code> , <code>standard</code>
<code>--s2p FILE</code>	<code>net</code>	write the channel's S-parameters (Touchstone)
<code>--s4p FILE</code>	<code>net</code>	write a pair's single-ended 4-port (Touchstone)
<code>--tree FILE</code>	<code>power</code> , <code>thermal</code>	power tree as JSON instead of the fields(<code>format</code>)
<code>--pdn RAIL</code>	<code>power</code>	add Z(f) of this rail
<code>--ambient °C</code>	<code>thermal</code>	ambient temperature (default 25)
<code>--airflow m/s</code>	<code>thermal</code>	forced air flow over the board
<code>--vertical</code>	<code>thermal</code>	the board stands vertically(natural convection)
<code>--h W/m²K</code>	<code>thermal</code>	surface coefficient per side instead of the convection correlations
<code>--via-cavity</code>	<code>net</code> , <code>check</code> , <code>report</code>	experimental: the via models' plane pairs as finite cavities (board edges, stitching vias, decoupling capacitors – estimated parasitics); slower
<code>--via-returns</code>	<code>net</code> , <code>check</code> , <code>report</code>	experimental: return groups per plane pair in the via models (a return via shorts only the pairs it joins; capacitors close supply/ground pairs)
<code>--out FILE</code>	<code>report</code> , <code>export</code>	where to write the report(default <code>analysis/kisi.json</code> next to the board) or the export's folder(default <code>analysis/export/<name></code>)
<code>--lane NAME</code>	<code>export</code>	the lane to export(a net of it)
<code>--bus NAME</code>	<code>export</code>	the bus to export, all its lanes(as <code>kisi interfaces</code> names it)
<code>--format LIST</code>	<code>export</code>	<code>lib</code> , <code>spice</code> , <code>xyce</code> , <code>ltspice</code> , <code>touchstone</code> , <code>kicad</code> (comma-separated; default all)
<code>--fmax GHZ</code>	<code>export</code>	top frequency of the ladders and the Touchstone grid (default 25 GHz, at least 1.25 × the bit rate); <code>profile</code> : 5 × Nyquist of the lane's profile
<code>--fmin GHZ</code>	<code>export</code>	start and step of the frequency grid (default f max / 500)
<code>--kicad-style STYLE</code>	<code>export</code>	<code>detailed</code> (one symbol per element, default) or <code>compact</code> (the lane as one symbol)
<code>--for VIEW</code>	<code>gaps</code>	only the gaps of one category: <code>impedance</code> , <code>channel</code> or <code>power</code>
<code>--formula</code>	field-solver commands	no field solver(formula impedances)
<code>--field-time S</code>	field-solver commands	time budget for new solves, default 120 s
<code>--field-all</code>	field-solver commands	all cross-sections, not only those of nets with a target
<code>--field-xt</code>	field-solver commands	also the coupling cross-sections of critical crosstalk victims

Assembly variants

`--variant NAME` (over `"variant"` in `<board>.kisi.json`) chooses which parts are fitted. When a board has variants – or forks where all alternative series parts are fitted – and none is chosen, `interfaces`, `check` and `report` stop with the list of variants(exit code 1) and `gaps` reports it; `power` and `thermal` warn and count the parts

as in the board file; the other commands do not depend on the fitted parts. `variant` in the output says which one was analysed. How variants are defined: [reference](#).

Field solver

`overview`, `net`, `check`, `crosstalk` and `report` run the 2D field solver (`kisi.mjs` only) for the cross-sections of nets with a target, the most routed length first, in worker threads, and stop starting new solves after `--field-time` seconds (default 120). The first run on a board can take minutes; results are cached in `~/.cache/kisi/field-cache.json` (`$XDG_CACHE_HOME/kisi`, Windows `%LOCALAPPDATA%\kisi`, or `$KISI_CACHE`), so later runs are fast and the coverage grows. `field_solver` in the output shows the coverage (`target_length_coverage_pct`); formula values fill the rest. `--formula` skips the solver; the native `kisi` always uses formulas and says so in `field_solver_note`.

Threads

`check` (and the `check` in `report` of the native build) computes its rows on up to 4 threads, fewer on machines with fewer cores; `KISI_THREADS=n` sets the number (1: no extra threads). The viewer's `check` uses Web Workers the same way. The result is identical to one thread.

Exit codes

0 success; 1 an error (unknown command or option, unreadable file, a lane command without a chosen variant), with `{"error": ...}` on stdout; the native `kisi` without arguments prints the help to stderr and exits with 2.

Using kisi with agents

Coding agents use kisi through its command line: `node kisi.mjs <command> <board>` prints JSON that states the models and assumptions behind every number. Two skills describe the work for agents; Claude Code installs them as a plugin, other agents can follow the same `SKILL.md` files.

The skills

Skill	What it does
kisi-prepare	runs <code>kisi gaps</code> , researches the missing values from datasheets (laminates $\epsilon_r/\tan\delta$, impedance targets, interface standards, regulator chain, load currents, thermal data, assembly variants, connector ends), writes them with sources to <code><board>.kisi.json</code> and checks the file with <code>kisi config-check</code>
kisi-si-pi-review	the SI/PI review: impedance per net class, channel checks against interface profiles, single nets, crosstalk, power tree, IR drop, PDN impedance and the temperature estimate (<code>overview</code> , <code>interfaces</code> , <code>check</code> , <code>net</code> , <code>crosstalk</code> , <code>power</code> , <code>thermal</code> , <code>report</code>), reported with the models' limits

A typical session: "prepare this board for kisi" (kisi-prepare), then "review its signal and power integrity" (kisi-si-pi-review). The worked example of a prepared board – the Seeed XIAO nRF54LM20A – is in [Preparing a board](#).

Setup

- Node.js ≥ 18 (nodejs.org, LTS).
- Claude Code: `/plugin marketplace add riehlies/kisi`, then `/plugin install kisi@kisi`.

The skills look for `kisi.mjs` in `$KISI_MJS`, then `~/.kisi/kisi.mjs`, and otherwise download the latest release into `~/.kisi/`:

```
curl -fsSL --create-dirs -o ~/.kisi/kisi.mjs
https://github.com/riehlies/kisi/releases/latest/download/kisi.mjs
# or
gh release download --repo riehlies/kisi --pattern kisi.mjs --dir ~/.kisi --clobber
```

(PowerShell: `Invoke-WebRequest <same URL> -OutFile $HOME\.kisi\kisi.mjs`.) A native `kisi` binary (`cargo install --path crates/kisi-cli`) accepts the same commands, without the field solver.

Notes for agents

- kisi reads the **saved** files; it does not talk to a running KiCad. Run it again after changes.
- Preparing needs no field solver: `gaps`, `config-check`, `config-template`, `interfaces` and `profiles` are fast. `overview`, `net`, `check`, `crosstalk` and `report` run the field solver; the first run on a board can take minutes (progress on stderr), later runs use the cache. `field_solver` in the output shows the coverage.
- Boards with assembly variants need one (`--variant` or `"variant"` in `kisi.json`); the lane commands stop with the list otherwise. Ask the user which one – never pick one silently.
- Every value written to `kisi.json` needs a source; estimates and assumptions are marked as such ([reference](#)).
- `kisi report` writes `analysis/kisi.json` next to the board, for file-based review tools (e.g. `kicad-happy`) and later sessions.
- With a tool that edits KiCad live (e.g. `Konnect`): change the copper, save, then run `kisi net` / `kisi check` again to confirm.

An MCP server and a KiCad plugin existed until October 2026 and were removed in favour of the CLI and the skills (they remain in the git history).