



Build a Versal FPGA from vendor data

AMD Versal Premium VP1002 · NFVI1369 · 1,369 balls

JITX JumpStart Kit 1 · Part 3 (component handling).

Parse the machine-readable pin file, generate the component, verify the generation — then wrap it in the circuit a board design consumes.

Choose the correct component ingestion flow

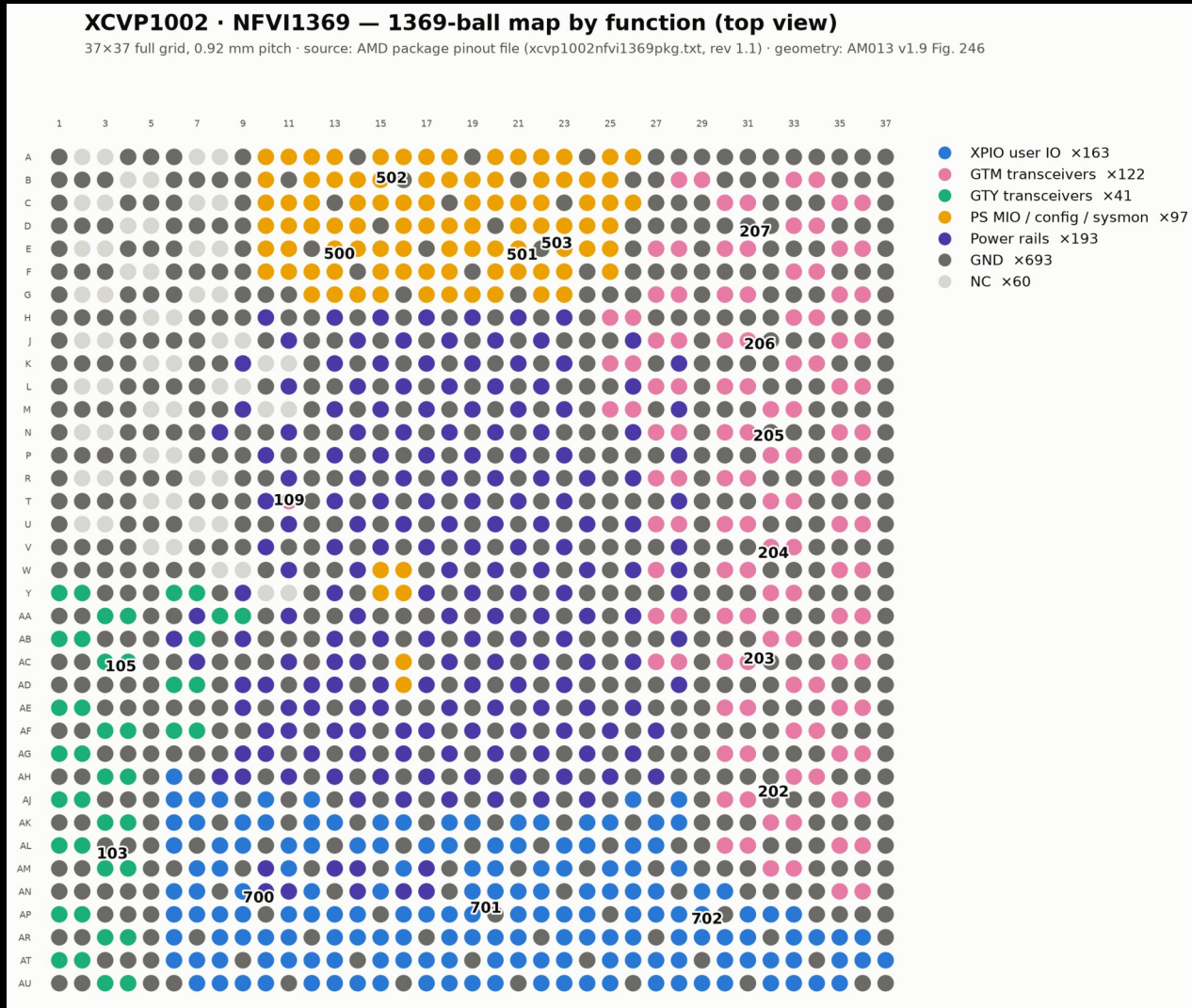
Part 3 teaches: the vendor's pin file becomes code. AMD publishes the ball map as a machine-readable ASCII file — names, balls, banks. That file is the only pin ground truth.

Parse, don't transcribe. A committed, stdlib-only generator reconciles the file to exactly 1,369 balls before any component code exists.

Generate, then verify the generation. Effort moves from typing to checking: inventory gates, hand-read spot-checks, geometry cross-checks, regeneration byte-identity.

Then lift the flat pin map into a circuit boundary. Power rails, ground domains, and GTM quads as JITX-native ports — what a board designer actually consumes.

One component, 1,369 balls, seven functions (simplified)



Rendered from the AMD pinout file (rev 1.1) · six GTM quads on the east column (banks 202–207) · XPIO south (700–702) · PS/config north (500–503)

What you build

tools/generate_pinout.py — committed stdlib-only generator. Parse → reconcile (1,369 = file footer, full 37×37) → emit. --report prints the inventory; --check proves the committed module regenerates byte-identically.

xcvp1002.py (generated) — one Port per ball: 432 unique pins under their exact AMD names, 30 repeated rails as indexed lists (GND ×689, VCCINT ×84, NC ×60), the ball map as zero-indexed coordinates, GTM quad groupings, symbol partitions.

landpattern.py — 37×37 BGA from AM013: body 35×35 mm (Package Dimensions), land Ø 0.51 mm (BGA Package Design Rules), public get_pad(row, column) adapter.

symbols.py — ~43 schematic boxes: one per bank / GT quad, rails chunked ≤ 64 pins per box; every port in exactly one box.

bundles.py + circuit.py — GTMQuad (4 lanes + 2 refclks) and XCVP1002Circuit: 31 Power rails, three ground domains (GND / GND_SMON / GND_SENSE), six GTM quad bundles wired through generated structural pin groupings.

Verification is the key

Reconcile before any code: parsed rows = file footer = 1,369; unique balls; full 37×37 grid.

Offline tests: 17 tests + 74 subtests on public jitx 4.2.2 — hand-read ball spot-checks across grid corners and every bank type, symbol coverage, mapping bijectivity, circuit rail roster.

Real builds: both designs (component viewer + wrapped circuit) build with status: ok against the JITX runtime.

Independent fan-out: a 13-agent review re-checked every one of the 1,369 balls against the AMD file — zero mismatches; rails, GTM, symbols, and conventions audits all pass.

Replay convergence: a fresh agent executed the runbook from its own text in an empty project — all four [HUMAN] gates, successful builds, and a ball map identical to the reference: 1,369/1,369 matching (name, row, col).

Geometry: A1 ↔ AU37 pad centers = $36 \times 0.92 = 33.12$ mm on both axes — AM013's D1/E1 exactly.

Run it

The runbook: `jumpstart-kits/js1-stackup-components/part3-versal-fpga/` — paste `JS1_Part3_Versal-FPGA-from-Pin-File_Runbook.md` into your agent and tell it to follow the file. Four [HUMAN] gates stop for you. Expect ~2–3 hours, most of it agent-autonomous.

The ground truth: download the `xcvp1002 NFVI1369` pinout file and `AM013` from `amd.com` during the run — vendor files are never committed.

The reference solution: `jitxexamples.jumpstart_kits.js1_stackup_components.versal_fpga` — generator, generated component, land pattern, symbols, circuit wrapper, tests, and two buildable designs.

Close the loop: "give me GTM quad 205 as a 4-lane bundle and tell me which power rails and bias pins it needs" — answered live from the built circuit.

Companion: `JS1_Part3_Reference_Companion.md` — the human-facing summary, inventory tables, and by-hand commands.