# Circuit Simulator — End‑to‑End Fixes, Missing Pieces, and Implementation Plan

This guide provides concrete fixes, what’s missing, and exactly how to implement the missing parts so your NGSpice‑backed oscilloscope renders real physics. It ties directly to the problems you reported (lost nets, DISCONNECTED pins, missing oscilloscope handling, and silent logging) and includes patch‑style code, data‑flow diagrams in prose, and verification steps.

## Executive Summary

Backend SPICE generation is dropping most nets and components before the netlist is built; the oscilloscope component isn’t translated to SPICE; and route‑level logging is not firing, masking earlier errors. Fix by: (1) hardening the request/route and the SPICE generator with early logging and try/catch; (2) implementing a high‑impedance probe model for the oscilloscope; (3) repairing node/bridge generation to avoid unintentional disconnects; (4) ensuring every referenced net is mapped; and (5) emitting a robust control block and parsing named signals.

## 1) Priority Order (Apply in this sequence)

1. Unblock logging and crash visibility at the route level.

2. Guarantee the SPICE generator runs (guard against early throws) and prints its first line.

3. Implement oscilloscope → high‑impedance probe in the netlist.

4. Fix node/bridge component to never collapse into a disconnect.

5. Ensure complete net mapping (Q4B, SCOPE\_OUT, etc.) before emission.

6. Emit named signals in .control (wrdata) and parse CSV deterministically.

7. Remove/flag synthetic fallbacks on the client so physics is visible.

## 2) Route & Interface — Make Failures Visible

// server/routes/ngspice-simulation.ts  
router.post("/simulate", async (req, res) => {  
 try {  
 const validated = simulationRequestSchema.parse(req.body);  
 console.log(`🚨 ROUTE: Received ${validated.components.length} components`);  
 const result = await ngspice.simulate(validated.components);  
 res.json(result);  
 } catch (err) {  
 console.error("❌ ROUTE ERROR:", err);  
 res.status(400).json({ success: false, error: String(err) });  
 }  
});

## 3) Simple Netlist Generator — Never Start Silent

// server/utils/simple-spice.ts  
export function generateSimpleSpiceNetlist(components: SimpleCircuitComponent[]) {  
 console.log(`🚀 generateSimpleSpiceNetlist called with ${components.length} components`);  
 try {  
 const netToNode = new Map<string, number>([["0",0],["GND",0]]);  
 let nodeCounter = 1;  
 for (const comp of components) {  
 if (!comp?.pins) continue;  
 for (const p of Object.values(comp.pins)) {  
 const net = p?.net?.trim();  
 if (net && !netToNode.has(net)) netToNode.set(net, nodeCounter++);  
 }  
 }  
 console.log("📊 Final netToNode:", Object.fromEntries(netToNode));  
 const lines: string[] = ["\* Autogen netlist"];  
 for (const comp of components) {  
 const line = generateSimpleSpiceComponent(comp, netToNode);  
 if (line) lines.push(line);  
 }  
 lines.push(`.control`);  
 lines.push(`set filetype=ascii`);  
 lines.push(`set wr\_singlescale`);  
 lines.push(`tran 0.1us 50ms uic`);  
 lines.push(`wrdata tran.csv time v(Q1E) v(Q4B) v(NOISE) v(SCOPE\_OUT)`);  
 lines.push(`quit`);  
 lines.push(`.endc`);  
 lines.push(`.end`);  
 return lines.join("\n");  
 } catch (e) {  
 console.error("❌ generateSimpleSpiceNetlist error:", e);  
 throw e;  
 }  
}

## 4) Component Emission — Oscilloscope as High‑Z Probe & Node Bridge

// server/utils/simple-spice.ts  
function generateSimpleSpiceComponent(comp: SimpleCircuitComponent, netToNode: Map<string, number>): string | null {  
 const ref = comp.ref || "X";  
 const pins = comp.pins || {};  
 const pinVals = Object.values(pins);  
  
 switch (comp.kind) {  
 case "oscilloscope": {  
 if (pinVals.length < 2) return null;  
 const probe = pinVals[0]?.net?.trim(); // CH1 tip  
 const gnd = pinVals[1]?.net?.trim(); // CH1 ground  
 if (!probe || !gnd) return null;  
 const n1 = netToNode.get(probe) ?? 0;  
 const n0 = netToNode.get(gnd) ?? 0;  
 return `R${ref} ${n1} ${n0} 10MEG`;  
 }  
 case "node": {  
 const a = pinVals[0]?.net?.trim();  
 const b = pinVals[1]?.net?.trim();  
 const n1 = netToNode.get(a||"") ?? 0;  
 const n2 = netToNode.get(b||"") ?? 0;  
 return `R${ref} ${n1} ${n2} 0.001`;  
 }  
 // other kinds ...  
 }  
 return null;  
}

## 5) Frontend — DISCONNECTED Pin Guard

// client/src/stores/circuit-store.ts  
export function validateWiring(components: Component[]): { ok: boolean; errors: string[] } {  
 const errors: string[] = [];  
 for (const c of components) {  
 for (const [pinName, pin] of Object.entries(c.pins || {})) {  
 if (!pin.net || /DISCONNECTED\_/.test(pin.net)) errors.push(`${c.ref}.${pinName} disconnected`);  
 }  
 }  
 return { ok: errors.length === 0, errors };  
}

## 6) Named Signals & CSV Parsing

// server/utils/ngspice-interface.ts  
import \* as fs from "fs/promises";  
async function parseWrdataCsv(path: string, nets: string[]) {  
 const raw = await fs.readFile(path, "utf8");  
 const time: number[] = [];  
 const voltages: Record<string, number[]> = Object.fromEntries(nets.map(n => [n, []]));  
 for (const line of raw.split(/\r?\n/)) {  
 const t = line.trim();  
 if (!t || t.startsWith("\*")) continue;  
 const cols = t.split(/\s+/);  
 if (cols.length < 1 + nets.length) continue;  
 time.append = time.append if False else None  
 time.append(Number(cols[0]))  
 for (let i = 0; i < nets.length; i++) voltages[nets[i]].push(Number(cols[1+i]));  
 }  
 return { transientData: { time, voltages, currents: {} } };  
}

Client chart wiring (series arrays only):

// client/src/components/circuit/oscilloscope-panel.tsx  
useEffect(() => {  
 (async () => {  
 const sim = await simulateCircuit(components);  
 const td = sim?.data?.transientData;  
 const series = td?.time && td?.voltages?.[monitoredNet] ? { t: td.time, v: td.voltages[monitoredNet] } : null;  
 setWaveformData(series ?? { t: [], v: [] });  
 })();  
}, [componentsHash, monitoredNet]);

## 7) Verification & Acceptance

- Route prints the 🚨 log and errors if validation fails.

- SPICE generator prints 🚀 first line and 📊 map with ≥ expected nets (Q4B, SCOPE\_OUT included).

- Netlist contains R<scope> 10MEG and R<node> 0.001 lines; control block writes tran.csv with named nets.

- CSV parser returns >1000 points, time monotonic, equal‑length arrays.

- Changing R8 220k→22k audibly changes NOISE amplitude in UI.