# NGSpice Oscilloscope — Full Fix & Verification Guide

This document gives exact, actionable fixes for the issues reported in your analysis, along with implementation steps, code snippets, and verification checklists. Apply these changes in order; each section maps to the specific file/line problems you listed.

## Contents

1. Priority Order (What to fix first)

2. Blocking NGSpice Errors (TRNOISE) — Replace & Alternatives

3. Interfaces & Field Propagation (model/type/polarity/comment)

4. TypeScript Compilation Errors (functions in blocks)

5. Unit Parsing & Precision (Ohms/Farads + formatting)

6. Netlist Generation Consistency (remove duplicates, one source of truth)

7. Deterministic & Random Stimuli (SIN/PULSE/PWL)

8. Evaluation & Tests (unit, integration, E2E)

9. Acceptance Criteria

10. Appendix: Ready-to-paste Code

## 1) Priority Order (What to fix first)

1. Fix NGSpice-blocking stimulus syntax (replace TRNOISE).

2. Fix TS compile error: move function declarations out of blocks.

3. Propagate model/type/polarity/comment fields to SimpleCircuitComponent.

4. Unify value parsing to one module; remove duplicates.

5. Add numeric formatting to avoid long floating strings.

6. Standardize netlist: values as clean SI or exponential; ensure single .end.

7. Rewire oscilloscope to read real transient arrays; disable synthetic math (if any left).

8. Add tests and a small verifier script to check transient data integrity.

## 2) Blocking NGSpice Errors (TRNOISE) — Replace & Alternatives

Problem: Your current NGSpice build rejects TRNOISE — it throws “parameter value out of range or the wrong type”. Use one of the following:

A) Deterministic stimulus (good for CI/verification):

# server/utils/simple-spice.ts (replace the two TRNOISE lines)  
lines.push(`VNOISE ${q1eNode} 0 SIN(0 0.001 1000)`); // 1 mV @ 1 kHz  
// If a legacy fallback line exists:  
lines.push("VNOISE 2 0 SIN(0 0.001 1000)");

B) Pseudo-random stimulus via PWL file (works on all ngspice builds):

# Generate noise.csv with (time,value) pairs externally (Gaussian 0-mean, e.g., 100 ns step).  
# Then in netlist:  
VNOISE Q1E 0 PWL FILE='noise.csv' REPEAT=YES

C) Upgrade path (optional): upgrade ngspice to a build that supports transient noise sources; then switch back to the supported syntax for your version.

## 3) Interfaces & Field Propagation (model/type/polarity/comment)

Add missing fields to SimpleCircuitComponent and propagate them during conversion so model selection and polarity-sensitive wiring work.

// server/utils/simple-spice.ts  
export interface SimpleCircuitComponent {  
 id: string;  
 ref: string;  
 kind: string;  
 model?: string; // ADDED  
 type?: string; // ADDED (if used upstream)  
 value?: string;  
 pins?: Record<string, { net: string; polarity?: "+" | "-"; comment?: string; }>; // ADDED  
 comment?: string; // ADDED  
}

// server/utils/ngspice-interface.ts (map fields through)  
const simpleComponents = components.map(c => ({  
 id: c.id,  
 ref: c.ref,  
 kind: c.kind,  
 model: c.model, // ADDED  
 type: c.type, // ADDED (if present)  
 value: c.value,  
 pins: c.pins, // retains polarity/comment if present  
 comment: c.comment // ADDED  
}));

## 4) TypeScript Compilation Errors (functions in blocks)

Move function declarations out of blocks or convert to const arrow functions.

// Before (inside a block):  
case "transistor": {  
 function pickBjtModel(component: { model?: string, ref?: string }) { ... }  
}  
  
// After: top-level  
function pickBjtModel(component: { model?: string, ref?: string }) {  
 const m = (component.model || "").toUpperCase();  
 if (m.includes("BC549")) return "BC549";  
 if (m.includes("BC337")) return "BC337";  
 if ((component.ref || "").toUpperCase() === "Q4") return "BC549";  
 return "BC337";  
}

## 5) Unit Parsing & Precision (Ohms/Farads + formatting)

Use a single parse module for SI values and introduce a formatter for netlist emission to avoid long floats like 0.0000099999999999.

// server/utils/units.ts  
export function parseSI(raw?: string): number { /\* (as implemented) \*/ }  
export const parseOhms = parseSI;  
export const parseFarads = parseSI;  
  
// Add a formatter:  
export function formatSpiceNumber(val: number, preferSI: boolean = true): string {  
 if (!Number.isFinite(val)) return "0";  
 if (!preferSI) return val.toExponential(6);  
 const abs = Math.abs(val);  
 if (abs >= 1e6) return (val/1e6).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "M";  
 if (abs >= 1e3) return (val/1e3).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "k";  
 if (abs >= 1) return val.toFixed(6).replace(/0+$/,"").replace(/\.$/,"");  
 if (abs >= 1e-3) return (val\*1e3).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "m";  
 if (abs >= 1e-6) return (val\*1e6).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "u";  
 if (abs >= 1e-9) return (val\*1e9).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "n";  
 return (val\*1e12).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "p";  
}

Emit netlist values via formatSpiceNumber (or exponential), not raw JS floats:

// Example for capacitors:  
const cVal = formatSpiceNumber(parseFarads(component.value));  
lines.push(`C${ref} ${n1} ${n2} ${cVal}`);

## 6) Netlist Generation Consistency (remove duplicates, one source of truth)

Delete duplicate parse functions from simple-spice.ts and spice-netlist.ts. Import from units.ts only, and standardize return types to numbers inside code; format only at the emission point.

// Remove in simple-spice.ts and spice-netlist.ts:  
function parseCapacitanceValue(...) { ... }  
function parseResistanceValue(...) { ... }  
  
// Keep only:  
import { parseOhms, parseFarads, formatSpiceNumber } from "../utils/units";

## 7) Deterministic & Random Stimuli (SIN/PULSE/PWL)

Recommended order: use SIN for CI (repeatable), offer PULSE for edge testing, and PWL FILE for realistic pseudo-random drive.

; Deterministic  
VNOISE Q1E 0 SIN(0 0.001 1000)  
  
; Pulse alternative  
VNOISE Q1E 0 PULSE(0 0.001 0 1n 1n 0.5u 1u)  
  
; PWL from file (random-looking, bandwidth set by time step)  
VNOISE Q1E 0 PWL FILE='noise.csv' REPEAT=YES

## 8) Evaluation & Tests (unit, integration, E2E)

Unit tests (Node/Jest):

- parseSI/parseFarads/parseOhms: '15kΩ' → 15000; '10µF' → 10e-6; exact matches.  
- formatSpiceNumber: 1e-5 → '10u' or '1e-5' (configurable).  
- pickBjtModel: 'BC549' → BC549; default Q4 → BC549; otherwise BC337.

Integration tests (server):

- Generate netlist → ensure single '.end' and no stripped '.tran'.  
- Run ngspice: no “parameter value out of range” errors.  
- Parse log: detect 'plotname: Transient Analysis'; time array length > 2; monotonic time.

End-to-End (client):

- Change R8 (220k → 22k): observe amplitude change at NOISE/SCOPE\_OUT.  
- Remove C1: output drops/changes as expected.  
- Toggle SIN↔PWL: visual change consistent with stimulus.  
- Cache: updates only when component hash changes; no stale waveforms.

## 9) Acceptance Criteria

- Build compiles without TS errors.

- NGSpice runs without syntax errors; transient plots present.

- Oscilloscope renders multi-point time series from NGSpice (not math).

- Component changes immediately reflect in waveform after a fresh sim.

- Netlist values are printed in clean SI/exponential (no long floats).

- Single source-of-truth parsing (no duplicates); consistent interfaces.

- Model/type/polarity/comment are preserved end-to-end.

## 10) Appendix: Ready-to-paste Code

A) SimpleCircuitComponent interface (expanded):

export interface SimpleCircuitComponent {  
 id: string;  
 ref: string;  
 kind: string;  
 model?: string;  
 type?: string;  
 value?: string;  
 pins?: Record<string, { net: string; polarity?: "+" | "-"; comment?: string; }>;  
 comment?: string;  
}

B) Field propagation in ngspice-interface.ts:

const simpleComponents = components.map(c => ({  
 id: c.id,  
 ref: c.ref,  
 kind: c.kind,  
 model: c.model,  
 type: c.type,  
 value: c.value,  
 pins: c.pins,  
 comment: c.comment  
}));

C) Unit helpers (units.ts):

export function parseSI(raw?: string): number { /\* your implementation \*/ }  
export const parseOhms = parseSI;  
export const parseFarads = parseSI;  
  
export function formatSpiceNumber(val: number, preferSI: boolean = true): string {  
 if (!Number.isFinite(val)) return "0";  
 if (!preferSI) return val.toExponential(6);  
 const abs = Math.abs(val);  
 if (abs >= 1e6) return (val/1e6).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "M";  
 if (abs >= 1e3) return (val/1e3).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "k";  
 if (abs >= 1) return val.toFixed(6).replace(/0+$/,"").replace(/\.$/,"");  
 if (abs >= 1e-3) return (val\*1e3).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "m";  
 if (abs >= 1e-6) return (val\*1e6).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "u";  
 if (abs >= 1e-9) return (val\*1e9).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "n";  
 return (val\*1e12).toFixed(6).replace(/0+$/,"").replace(/\.$/,"") + "p";  
}

D) Example netlist stimulus options:

VNOISE Q1E 0 SIN(0 0.001 1000)  
\* or  
VNOISE Q1E 0 PULSE(0 0.001 0 1n 1n 0.5u 1u)  
\* or  
VNOISE Q1E 0 PWL FILE='noise.csv' REPEAT=YES