# Oscilloscope Flat Line — Complete Fix & Implementation Guide

This guide explains exactly what is wrong, how to fix it, and what to implement so the oscilloscope renders real NGSpice transient waveforms. It includes file-by-file patches, TypeScript interfaces, client/server wiring, performance-minded sampling, and verification steps.

## Executive Summary

Root cause: NGSpice produces full transient time-series, but the server returns only DC (operating-point) values, and the client renders those static values repeatedly. Fix: parse and return transient arrays on the server; consume them on the client; render time-series (not single numbers) in the oscilloscope.

## What’s Wrong (from your analysis)

- Transient time-series is generated by NGSpice but discarded by the server parser.

- NgSpiceResult interface lacks a transientData field, so data cannot propagate to the client.

- simulateCircuitProfessional()/getSignalAtNode() return only scalar DC values.

- Oscilloscope panel appends the same scalar value each frame → a flat line.

- Net-to-node mapping is underused in transient sampling; channel subtraction (pos–neg) is not based on aligned arrays.

## Implementation Overview (What to Change)

- Server: Call parseTransientData(stdout) and include its output in NgSpiceResult.data.transientData.

- Server: Extend NgSpiceResult interface with a transientData field (time, voltages, currents).

- Client: Add helpers to fetch a complete series {t, v} for a given net using netToNodeMap.

- Client: Update oscilloscope panel to render full series (with decimation), not scalar values.

- Client: Implement differential channel (pos–neg) by subtracting arrays element-wise.

- Add tests and a verifier to ensure time is monotonic and arrays are aligned.

## Server Patches — ngspice-interface.ts

1) Extend interfaces to carry transient data:

// server/utils/ngspice-interface.ts  
  
export interface NgSpiceTransient {  
 time: number[]; // seconds  
 voltages: Record<string, number[]>; // key = node number as string  
 currents: Record<string, number[]>; // key = element name  
}  
  
export interface NgSpiceResult {  
 success: boolean;  
 error?: string;  
 data?: {  
 operatingPoint: Record<string, number>; // e.g., { v\_1: 5, v\_2: 0 }  
 netToNodeMap?: Record<string, number>; // e.g., { 'SCOPE\_OUT': 3 }  
 transientData?: NgSpiceTransient; // <-- NEW  
 };  
}

2) Call parseTransientData and attach output:

// Inside the method that runs ngspice and captures stdout  
const opResult = this.parseNgSpiceOutput(stdout); // existing DC parse  
const tran = this.parseTransientData(stdout); // <-- ADD THIS  
  
const data: NgSpiceResult['data'] = {  
 operatingPoint: opResult.operatingPoint || {},  
 netToNodeMap: opResult.netToNodeMap || {}  
};  
  
if (tran && tran.time?.length > 2) {  
 data.transientData = tran;  
}  
  
return { success: true, data };

3) Ensure parseTransientData returns aligned arrays:

// server/utils/ngspice-interface.ts (already implemented per your analysis)  
// Ensure it returns: { time: number[], voltages: Record<string, number[]>, currents: Record<string, number[]> }  
// and that every voltages[key].length === time.length

## Client Utilities — circuit-simulation.ts

Add helpers to convert NgSpiceResult to time-series for any net name:

// client/src/utils/circuit-simulation.ts  
  
export interface Series {  
 t: number[]; // seconds  
 v: number[]; // volts  
}  
  
export function getSeriesForNet(result: any, netName: string): Series | null {  
 const td = result?.data?.transientData;  
 const map = result?.data?.netToNodeMap;  
 if (!td || !map || !netName) return null;  
  
 const node = map[netName];  
 if (node == null) return null;  
  
 const key = String(node);  
 const time = td.time || [];  
 const volts = td.voltages?.[key] || [];  
  
 if (!time.length || volts.length !== time.length) return null;  
 return { t: time, v: volts };  
}  
  
// Differential channel: v = v(pos) - v(neg)  
export function getDifferentialSeries(result: any, posNet: string, negNet?: string): Series | null {  
 const pos = getSeriesForNet(result, posNet);  
 if (!pos) return null;  
 if (!negNet) return pos;  
  
 const neg = getSeriesForNet(result, negNet);  
 if (!neg || neg.t.length !== pos.t.length) return pos; // fallback single-ended  
  
 const v = pos.v.map((vp, i) => vp - (neg.v[i] ?? 0));  
 return { t: pos.t, v };  
}  
  
// Visual downsampling (Largest-Triangle-Three-Buckets)  
export function lttbDownsample(series: Series, threshold = 1200): Series {  
 const { t, v } = series;  
 const n = t.length;  
 if (n <= threshold) return series;  
  
 const sampledT: number[] = [];  
 const sampledV: number[] = [];  
  
 let bucketSize = (n - 2) / (threshold - 2);  
 let a = 0;  
  
 sampledT.push(t[0]); sampledV.push(v[0]);  
  
 for (let i = 0; i < threshold - 2; i++) {  
 const start = Math.floor((i + 1) \* bucketSize) + 1;  
 const end = Math.floor((i + 2) \* bucketSize) + 1;  
 const avgRangeStart = Math.max(start, 0);  
 const avgRangeEnd = Math.min(end, n);  
 let avgT = 0, avgV = 0, avgN = Math.max(avgRangeEnd - avgRangeStart, 1);  
 for (let j = avgRangeStart; j < avgRangeEnd; j++) { avgT += t[j]; avgV += v[j]; }  
 avgT /= avgN; avgV /= avgN;  
  
 const rangeStart = Math.floor(i \* bucketSize) + 1;  
 const rangeEnd = Math.floor((i + 1) \* bucketSize) + 1;  
  
 let maxArea = -1, maxIndex = rangeStart;  
 for (let j = rangeStart; j < Math.min(rangeEnd, n); j++) {  
 const area = Math.abs((t[a] - avgT) \* (v[j] - v[a]) - (t[a] - t[j]) \* (avgV - v[a]));  
 if (area > maxArea) { maxArea = area; maxIndex = j; }  
 }  
 sampledT.push(t[maxIndex]); sampledV.push(v[maxIndex]);  
 a = maxIndex;  
 }  
  
 sampledT.push(t[n - 1]); sampledV.push(v[n - 1]);  
 return { t: sampledT, v: sampledV };  
}

## Oscilloscope Panel — oscilloscope-panel.tsx

Replace scalar polling with time-series rendering:

// client/src/components/circuit/oscilloscope-panel.tsx  
  
import { getDifferentialSeries, lttbDownsample } from "@/utils/circuit-simulation";  
  
async function refreshWaveform() {  
 const sim = await simulateCircuitProfessional(components); // MUST return NgSpiceResult with transientData  
 const series = getDifferentialSeries(sim, monitoredNetPos, monitoredNetNeg); // neg optional  
 if (!series) { setWaveformData({ t: [], v: [] }); return; }  
  
 const ds = lttbDownsample(series, 1200); // render-friendly  
 setWaveformData(ds);  
}  
  
// In a useEffect or on-demand refresh:  
useEffect(() => { refreshWaveform(); }, [componentsHash, monitoredNetPos, monitoredNetNeg]);

Render the arrays with your charting lib (Canvas/WebGL/SVG). Now you set full arrays, not append a single scalar at each tick.

## simulateCircuitProfessional Contract

Ensure the function returns NgSpiceResult with transientData:

type NgSpiceResult = {  
 success: boolean;  
 data?: {  
 operatingPoint: Record<string, number>;  
 netToNodeMap?: Record<string, number>;  
 transientData?: {  
 time: number[];  
 voltages: Record<string, number[]>;  
 currents: Record<string, number[]>;  
 };  
 };  
};

## Diagnostics & Verification

Add minimal logging to prove the pipeline is correct:

// server/utils/ngspice-interface.ts  
console.info("Transient points:", data.transientData?.time?.length || 0);  
if (data.transientData) {  
 const anyNode = Object.keys(data.transientData.voltages)[0];  
 console.info("Example node length:", data.transientData.voltages[anyNode]?.length);  
}  
  
// client/src/utils/circuit-simulation.ts  
const ok = Array.isArray(series.t) && series.t.length > 2 &&  
 Array.isArray(series.v) && series.v.length === series.t.length;  
if (!ok) console.warn("Series invalid for", netName, series);

Add tests:

Unit:  
- parseTransientData returns time[].length > 10 and voltages keyed by node with equal length.  
- getSeriesForNet finds node by name from netToNodeMap and returns aligned arrays.  
- lttbDownsample reduces points to threshold while preserving endpoints.  
  
Integration:  
- netlist with SIN(0 1m 1k); expect transient time monotonic and non-constant voltages.  
- server returns NgSpiceResult.data.transientData populated.  
  
E2E:  
- Change a component (e.g., R8 220k→22k): waveform amplitude/frequency content changes on next run.  
- Switch monitored net; oscilloscope displays new series.

## Acceptance Criteria

- NgSpiceResult includes data.transientData with time[] and voltages{} arrays of equal length (> 100 points).

- Oscilloscope renders a non-flat waveform that changes in response to component edits.

- No synthetic noise/math is added to displayed values (unless behind a dev flag).

- Time series downsampling is applied for performance without distorting endpoints.

- Monotonic time verified; series alignment verified; differential mode works (pos–neg).

## Fallback Behavior

- If transientData is missing, display a clear banner “No transient data – showing DC only” and render a flat line intentionally.  
- If a net name is not in netToNodeMap, surface an error and suggest available nets.  
- If arrays are extremely large (>5e6 points), force stronger decimation or cap duration window.