# Coupling Simulation and Hardware for Interactive Circuit Debugging

Evan Strasnick
estrasni@stanford.edu
Stanford University
Stanford, CA

Maneesh Agrawala
agrawala@cs.stanford.edu
Stanford University
Stanford, CA

Sean Follmer
sfollmer@stanford.edu
Stanford University
Stanford, CA

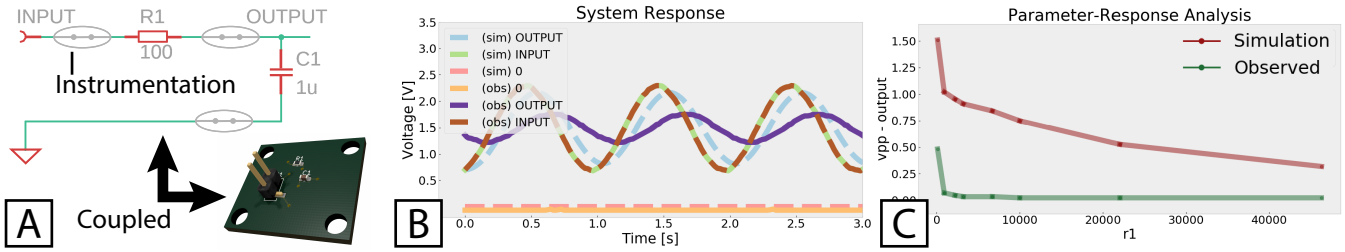


**Figure 1: Simpoint uses a simulated model coupled with the designer's physical circuit to aid in analog circuit debugging. A) A simple RC filter stage instrumented for use with Simpoint. Here we show the circuit schematic and the hardware, kept synchronized during debugging. B) Simpoint measures live signals from the hardware and juxtaposes them alongside simulated responses to the same inputs. Here, the designer observes that the output signal in their physical circuit (purple) exhibits greater attenuation than its simulated counterpart (light blue), suggesting either excess resistance or excess capacitance. C) Simpoint can also programmatically modify both signals and component parameters of the physical circuit to enable automated parameter-response analyses typical of simulation. Here, Simpoint compares how the peak-to-peak voltage of the output signal varies as a function of the resistor R1 both in the simulation and on the hardware. Given the uniform increase in attenuation across the resistor's range, excess capacitance is a more likely issue, and decreasing the resistance is unlikely to correct the behavior.**

## ABSTRACT

Simulation offers many advantages when designing analog circuits. Designers can explore alternatives quickly, without added cost or risk of hardware faults. However, it is challenging to use simulation as an aid during interactive debugging of physical circuits, due to difficulties in comparing simulated analyses with hardware measurements. Designers must continually configure simulations to match the state of the physical circuit (e.g. capturing sensor inputs), and must manually rework the hardware to replicate changes or analyses performed in simulation. We propose techniques leveraging instrumentation and programmable test hardware to create a tight coupling between a physical circuit and its simulated model. Bridging these representations helps designers to compare simulated and measured behaviors, and to quickly perform analytical techniques on hardware (e.g. parameter-response analysis) that are typically cumbersome outside of simulation. We implement these techniques in a prototype and show how it aids in efficiently debugging a variety of analog circuits.

## CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; • **Hardware** → *PCB design and layout*; **Hardware test**.

## KEYWORDS

debugging, circuit, PCB, simulation, analysis, testing

**ACM Reference Format:**
Evan Strasnick, Maneesh Agrawala, and Sean Follmer. 2021. Coupling Simulation and Hardware for Interactive Circuit Debugging. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, May 08–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456


*CHI '21, May 08–13, 2021, Yokohama, Japan*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Simulation is a valuable tool in analog circuit design. Designers of analog circuits ("designers") use simulations to explore potential designs quickly and at low cost. Before testing and debugging a physical prototype, designers often first verify the design in simulation.

However, while simulation is most frequently used in early design stages, it can also serve as a useful tool in the debugging of physical circuits. We highlight two benefits: First, it provides a

reference to help the designer understand *what correct behavior looks like*. As the designer tests different inputs or tries exploratory changes to the physical circuit, simulation offers a point of comparison with a theoretical response. Second, it offers efficient ways to *explore alternatives*. As modifications are easy to perform in the simulation, the designer can quickly investigate the effects of a given parameter variation. Simulations can further *automate* the exploration of alternatives, in the form of *parameter-response analyses* that reveal patterns of behavior. For example, sensitivity analysis can summarize how individual circuit parameters contribute to behaviors of interest [11, 21], helping the designer to reason about potential sources of error and their solutions.

Despite these advantages, barriers in comparing simulated analyses with hardware measurements make it cumbersome to use simulation as an aid when debugging physical circuits. To enable comparison between these representations, the designer must configure simulations that reflect the current state of the device under test (DUT), replicate the same inputs observed on the hardware, and render output that aligns with that of an inspection tool (e.g. a triggered oscilloscope trace). Each time the designer alters the hardware, environment, or inputs, they must reproduce that change on the simulation in turn. Similarly, the designer can explore a change easily in simulation, but to evaluate the equivalent response on the hardware requires reworking the physical circuit. While simulations can run parameter-response analyses automatically, it is a slow, error prone process to manually replace on-board components and carry out similar analyses on hardware.

The central contribution of this work is a set of techniques to aid in debugging by creating a *tight coupling between a live circuit and its simulated model*. Specifically, we present a prototype of a hardware-software PCB debugging tool, Simpoint, which facilitates comparison between the two representations both by updating the simulation in response to changes on the physical circuit, and by modifying the hardware to mirror programmatic changes in the simulation.

*Mirroring hardware state in the simulation* enables the designer to reference simulated counterparts of each live signal measured on their physical circuit, even after changes to the hardware or its inputs. Simpoint injects measured inputs from the hardware into the simulation, and automatically juxtaposes equivalent simulated responses with measured outputs, as in Figure 1.B. The designer can also select signals produced by the simulation and inject them back into the physical circuit.

*Mirroring simulation changes on the hardware* enables the designer to compare patterns of behavior in their physical circuit while exploring them in simulation. Using debugging hardware and software that can programmatically manipulate component parameters on instrumented PCBs, designers can tune passive component parameters up and down, or can utilize automated parameter-response analyses mirroring those available in simulation, as in Figure 1.C. As a result, the designer can compare their physical and simulated circuits with questions such as, "How does my circuit's functionality change as I vary parameter X?" or "How can I adjust my circuit to tune the response at point Y?" They can run these analyses during hands-on debugging, or can articulate expected behaviors using a specification language to enable automated testing of functional relationships (e.g. "verify that the rise time at node N1 increases with greater values of capacitor C1").

Our Simpoint prototype was built on top of the Pinpoint PCB debugging tool, described by Strasnick et al. [23]. Pinpoint simplifies the process of interfacing a PCB with debugging software. Briefly, given a schematic and PCB layout in a conventional CAD tool (e.g. EAGLE [4]), Pinpoint automatically inserts test points ("jumper pads") into the traces on the board, and then generates designs for a physical test jig that interfaces these points with testing hardware, called the "Control Board." At any instrumented point, the jumper pad allows the designer to probe the signal, inject a generated or recorded signal, and open or close connectivity at that juncture. Simpoint builds on this infrastructure, adding new hardware and software components (detailed in Section 3) that leverage the advantages of simulation and mirror debugging operations across the two representations. Like Pinpoint, we designed Simpoint to support analog circuit designers with moderate experience but without access to industry-level specialized testing equipment. This includes students, independent designers, hobbyists, and researchers like ourselves.

## 2 RELATED WORK

In addition to simulation, designers make use of a diverse set of inspection tools across the various stages of the design process. During prototyping and refinement, manual tools such as oscilloscopes, multimeters, and logic analyzers provide flexible measurements of signal properties with minimal overhead. For later stages of design and production, automated test platforms can more efficiently perform verification and validation on a target circuit. Other in-circuit testing methods leverage Built-In Self Test (BIST) capabilities designed into components, such as the widely adopted JTAG standard [12]. Below we describe ongoing research efforts towards improving this set of tools for both physical and simulated circuits.

### 2.1 Augmenting Inspection

Driven by the rise of "maker" cultures, renewed interest in end-user circuit debugging has led to the development of low-threshold tools for augmented inspection of circuits. Prototyping tools such as the Toastboard [16], CurrentViz [28], and Heimdall [13] offer ways to quickly inspect properties of breadboarded circuits, while tools such as Pinpoint [23] aid in generating test interfaces to debug PCBs in software. Simpoint adds to these techniques the means to inspect both the physical circuit and its simulation in parallel.

Simpoint also aims to facilitate inspection of functional relationships through automated parameter-response analysis. Beyond simulation, efforts in other domains to highlight relationships between parameters and outcomes have utilized alternative-choice-based approaches, such as Design Galleries [15], Parallel Paths [25], and subjunctive interfaces [15]. The Juxtapose interface applies these techniques in a tool for parallel tuning of software parameters while monitoring live output [10]. Simpoint applies similar principles to the task of hardware debugging by sampling points from a parameter space and summarizing live responses.

## 2.2 Simulation

Designers typically utilize simulation techniques early in the design process to verify behaviors before committing to a design. Relative to a physical prototype, simulations are both faster and cheaper to modify, as they do not require parts, fabrication time, or manual assembly. Modern computer-aided design (CAD) tools often have built-in capabilities for common analog simulators such as SPICE [17], with other dedicated simulation packages available for specialized tasks including filter design, power integrity analysis, or noise modeling. Simulation can also aid in automating the generation of test patterns for verification [7].

Simulating analog behaviors requires a high degree of precision, as analog circuits represent continuous transforms on continuous signals, where errors can compound over multiple stages. (In contrast, small deviations in digital circuits can be corrected back to high/low values at each stage.) Analog circuits are thus more sensitive to component parameter variations, noise, and other factors not explicitly captured by the circuit model, resulting in discrepancies between the simulated circuit and measured behaviors. Simpoint helps by highlighting where these errors occur (by comparing measured and simulated behaviors) and by comparing how potential circuit modifications can reduce these errors.

We draw attention to two applications of simulation with particular relevance to parameter-response analysis: *fault modeling* and *sensitivity analysis*:

Fault modeling refers to the practice of modeling a hypothetical fault and observing how error behaviors propagate. It sees usage most often in digital verification, as clear input-output chaining and discrete transfer functions make it straightforward to propagate error behaviors through the system. Fault modeling tools for analog systems face additional challenges, as continuous signals are harder to match against a set of models and any inaccuracies in simulation quickly compound through propagation [15]. We do not characterize Simpoint as a fault modeling tool, but designers can use it to temporarily inject a fault into their circuit and observe both measured and physical responses (see example A1 in Section 5).

Sensitivity analysis describes techniques that model how properties of circuit behavior vary with changes to a given design parameter. Sensitivity analysis can be an effective alternative to trial-and-error when optimizing designs, or when characterizing a design space for which concise, closed-form descriptions are not available. Sensitivity-based techniques have also been utilized in fault modeling approaches to infer parameter variations that most plausibly explain observed behaviors [11, 21]. Simpoint's parameter-response analyses can be considered coarse empirical analogues of sensitivity analysis, used to characterize parameter-response relationships without assuming an underlying model.

## 2.3 Bridging Hardware and Simulation

Simpoint draws inspiration from an established engineering practice known as hardware-in-the-loop (HWIL) simulation. HWIL refers to "breaking out" parts of a simulated environment to pass instead through physical hardware components, often to avoid describing those elements mathematically [5]. For example, when designing control systems in simulation, designers may use a physical actuator connected to simulated logic rather than try to capture the actuator's dynamics. With the rise of the FPGA as a platform for prototyping digital systems, recent work has developed tools to use FPGAs and other programmable systems-on-a-chip in a HWIL fashion to accelerate computationally expensive stages of simulation [7]. While Simpoint shares with HWIL techniques the underlying insight of coupling simulation with hardware, they address fundamentally different challenges. HWIL techniques use physical components to improve the efficiency and accuracy with which a designer can develop a simulation. Simpoint instead integrates simulated models and programmable hardware into the debugging of physical circuits to facilitate helpful analyses.

Several tools have also made efforts to bridge the gap between analog debugging and CAD tools. CircuitSense [27] updates a CAD representation when components are placed on a physical breadboard, while BoardLab [9] helps the designer to locate nodes in a schematic by probing sites on a physical PCB. Tools such as VirtualComponent [14] allow modifications made in a CAD representation to update hardware in real time. However, efforts to directly link live hardware with circuit simulations have focused mainly on digital logic systems. For example, FPGA-assisted co-simulation enables designers to run a simulation in parallel with synthesized digital logic on a programmable device [3, 26]. Co-simulation is therefore useful in validating the digital implementation for an analog process by contrasting behaviors between the two representations. In terms of debugging, this paradigm can further combine the strengths of emulation and simulation by running execution on hardware (fast execution, but low visibility) until an error is reached, and then configuring an appropriate simulation for debugging (high visibility, but slow), as in StateMover [6]. However, co-simulation is less frequently used for evaluating analog hardware, as there is no such analog-digital difference in the design and algorithm. Thus, a single simulation gives the expected behaviors of the system in a 1-to-1 fashion. Simpoint instead supports the designer in frequently making modifications and exploring alternatives while debugging in real time, making it necessary to continually revise and re-run simulations. For this reason, we find utility in coupling simulations to offer comparisons with expected behaviors as modifications occur.

# 3 TOOL OVERVIEW

Simpoint consists of testing hardware that measures and modifies the designer's PCB, as well as a software interface (Figure 2) through which the author performs debugging. We introduce its novel capabilities at a high level below, and then in Section 4 offer further details and describe the process of instrumenting a PCB for use with Simpoint. In Section 5, we demonstrate Simpoint's usage in a variety of example debugging scenarios.

## 3.1 Comparing Behaviors Across Simulation and Hardware

Simpoint's primary function is the simultaneous capture and comparison between a physical circuit and its simulation. The designer can measure signals throughout the hardware, while Simpoint provides simulated counterparts of the same responses, even as the circuit or its inputs change. For example, upon finding that the

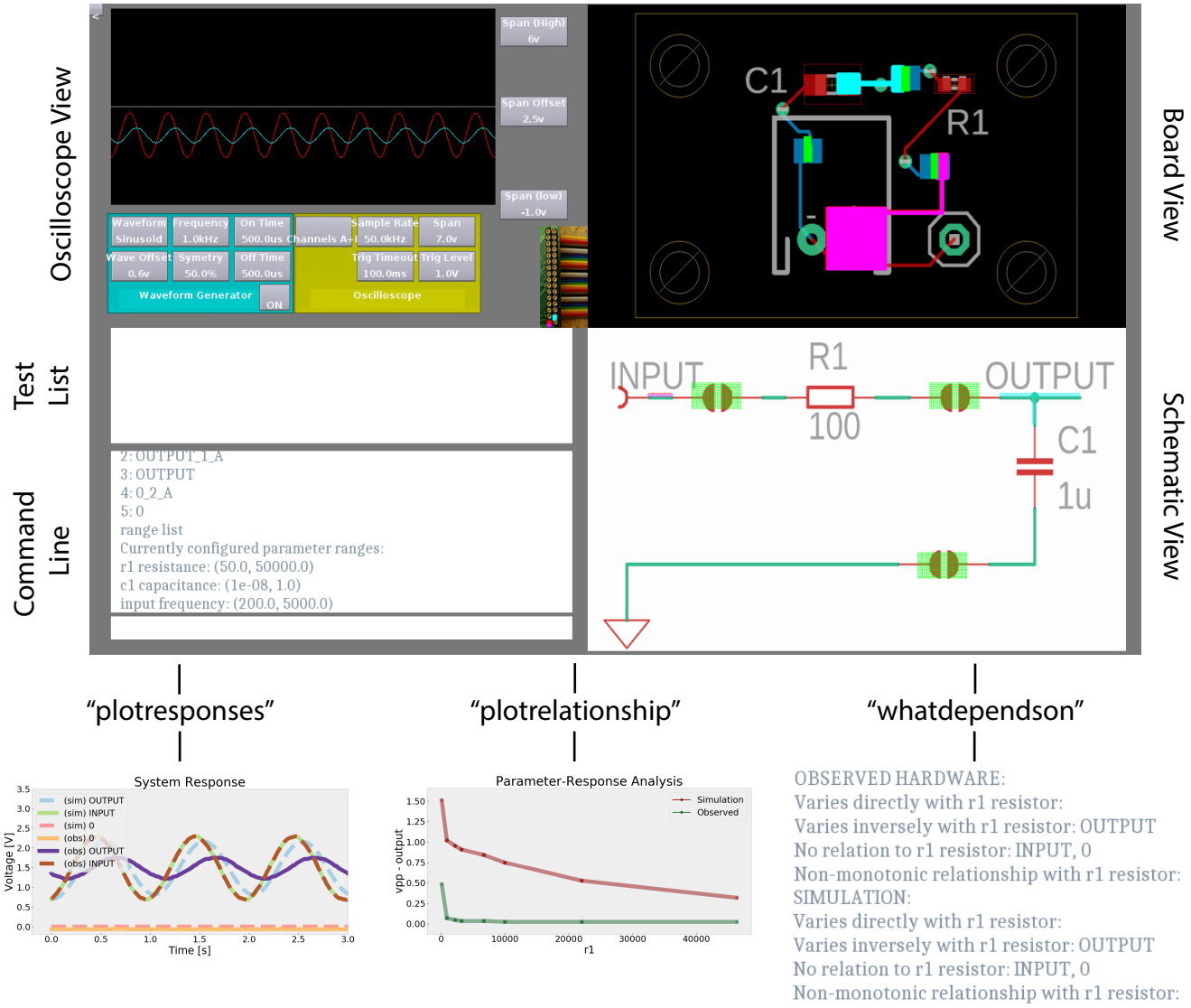


**Figure 2: Simpoint's software interface, built upon Strasnick et al.'s original tool [23]. It includes a traditional oscilloscope view, board and schematic views that facilitate probing or injecting signals across the circuit, and a command line interface. Plotted responses and relationships pop up in interactive visualizations, while summary analyses appear as command line output.**

output of their circuit does not match their expectations, a designer might interactively compare responses backwards along the signal chain to find where behaviors in the hardware first deviate from the simulation.

For circuits with external inputs, the user can designate any hardware signal as input to the simulation. Starting with this input, Simpoint in turn models the remainder of responses throughout the simulated circuit, and juxtaposes them in an interactive visualization as shown in Figure 1.B. Simulated responses can in turn be injected back into the physical circuit as signals, e.g. allowing the designer to bypass a faulty stage and evaluate outputs given simulated inputs. The simulation is always kept synchronized to the state of the hardware for changes made through the Simpoint interface, such as temporarily isolating an element or updating a component value.

## 3.2 Comparing Across Alternatives

Simpoint conversely can programmatically perform certain modifications on the physical hardware to mirror changes in simulation. This capability enables the designer to compare responses of the physical circuit as alternative designs are explored in the simulation. These explorations can be automated in the form of parameter-response analyses, which quickly show the expected circuit behaviors as a parameter is varied across its range. An example

output is shown in Figure 1.C. Analogous to a manual process of repeatedly removing on-board components, replacing them with different-valued substitutes, and retaking measurements, Simpoint instead performs these operations programmatically, drawing replacement components from a modular bank. Simpoint also utilizes its arbitrary waveform generator (AWG) to vary signal properties such as voltage and frequency as parameters in these analyses. By comparing how responses in both the physical circuit and simulation respond to changes in the parameter, the designer can observe differences in functional relationships between the two representations. For example, a designer might find that leakage currents cause voltages to decrease on the physical circuit (but not the simulation) when varying resistance.

Beyond individual parameter-response analyses, Simpoint also enables the designer to run "circuit-wide" parameter-response analyses, summarizing trends across all possible parameters or all possible responses. These analyses help the designer to quickly answer questions such as "Which signals in my circuit depend upon parameter X?" or "Which parameters significantly affect the response at Y?" Again, by comparing these results from the simulated and physical circuits, Simpoint can highlight functional differences between them, such as a dependency that exists in the hardware but not the simulation.

### 3.3 Functional Testing

Finally, after using hardware-simulation comparison and parameter-response analysis to debug circuit behaviors, Simpoint can codify correct behaviors as specifications to automate functional testing. Once authored, functional tests can alert the designer in the future when fabricated units fail to exhibit a specified behavior. Simpoint offers a specification language that facilitates the expression of expected signal properties in terms of circuit parameters. For example, a designer might specify that the peak-to-peak output from a low-pass RC filtering stage should decrease with the frequency of the input as well as with the values of R and C. Given parameter ranges, test signals, and tolerances, Simpoint automatically performs the necessary parameter modifications, takes measurements, and reports when specified relationships fail to hold true. These specifications are evaluated on both the simulated circuit and the hardware, offering insights into whether an error is likely to be an error in design versus an error in construction. Simpoint can even automatically generate and evaluate specifications that check for common issues, such as power supply noise. The details of Simpoint's specification language are included in Appendix A.

## 4 IMPLEMENTATION AND TECHNICAL CONSIDERATIONS

Three components enable Simpoint's novel functionality: First, testing hardware can programmatically substitute components on the user's board with components in an modular external bank, e.g. for automated parameter-response analysis. Second, a coupled SPICE simulation mirrors the state of the designer's circuit, measurement settings (e.g. sampling rate), and input signals. Finally, a specification language enables the authoring of functional tests for expected values and expected relationships.

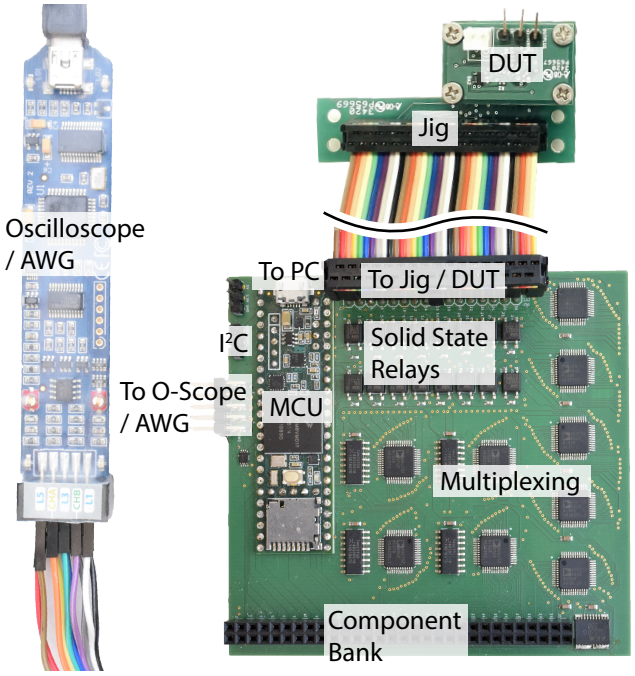


**Figure 3: Simpoint's testing hardware connects to the designer's jig to inspect and modify their PCB (DUT). Solid state relays programmatically toggle connections in the physical circuit, and analog multiplexers route signals to the oscilloscope and AWG, as well as across components in the modular component bank. The software interface controls the hardware over serial connection to the microcontroller, and exposed $I^2C$ pins allow for communication with external devices.**

### 4.1 Testing Hardware

To connect the designer's PCB with testing hardware, Simpoint extends Strasnick et al.'s [23] approach to automated PCB instrumentation. Simpoint's testing hardware, shown in Figure 3, includes analog multiplexers [8] for routing instrumented signals to a USB oscilloscope, solid-state relays [24] for bridging jumper pad connections, and an on-board microcontroller [19] for communication with the software interface via serial. The primary addition to Simpoint's hardware is the usage of additional multiplexing to connect instrumented sites on the board with external, two-terminal components in a modular bank. The multiplexing configuration is shown in Figure 4. Simpoint approximates the modification of a fixed component parameter by replacing the on-board component with a similar external component differing solely in the target parameter. Any two points from the 32 instrumented sites in the designer's circuit can be connected to either end of the 30 rows in the component bank, as well as dedicated rows for short and open connections.

The component bank is implemented as a 32x2 header row to which the designer connects external components such as passives (resistors, capacitors, etc.), integrated circuit I/O, programmable

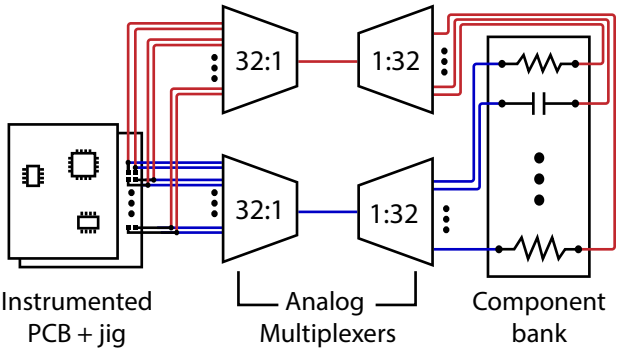


**Figure 4: Simpoint uses analog multiplexers to arbitrarily connect sites from the physical circuit to a row in the modular component bank. Color is used only to highlight the two "halves" of the pathway. Multiplexing is also used in routing instrumented sites to the oscilloscope and AWG (not pictured).**

components, or external measurement devices. The designer specifies the types and values of the connected components to a config file, so that Simpoint can automatically choose from available components during parameter sweep operations. The testing hardware additionally exposes $I^2C$ communication pins, to be used with any $I^2C$-enabled external devices. Using these pins, the designer can modify Simpoint's on-board microcontroller code to additionally configure external components (e.g. adjust the value of a programmable resistor) as a part of test routines.

## 4.2 Coupled SPICE Simulation

As part of its automated instrumentation process, Simpoint additionally outputs a SPICE simulation file describing the instrumented circuit and all included component models. The jumper pads added during instrumentation are modeled as externally controlled switches corresponding to the solid-state relays that bridge them, using the measured on-state resistance (.8 Ω), off-state resistance (40 MΩ), and output capacitance (.17 nF) [24].

Simpoint's test software uses a Python library [20] to interface with the Ngspice simulation package [1]. When the designer begins debugging, Simpoint loads the generated SPICE file for the circuit's initial state along with the other board and schematic metadata. Debugging operations performed on the physical circuit update the simulation as well: When the designer toggles the state of a jumper pad, the corresponding component model is switched between its on and off state. When the arbitrary waveform generator (AWG) is set at a target location, an externally-controlled voltage source is placed at the same point in the simulation, pulling values directly from the output of the (physical) AWG. Modifications to component parameters (such as those performed during parameter-response analysis) update the corresponding component parameter in the simulation. The user can also manually add/delete elements or modify element parameters, for situations in which they make changes to the hardware outside of the Simpoint software.

For Simpoint to produce simulated responses to signals measured on the physical circuit, the designer first designates a signal as the "input". Observed traces from this signal are treated as "ground-truth" – an externally controlled voltage source is placed on the corresponding node in the simulation, with the values of the observed trace as its output. To propagate this signal, Simpoint then runs a transient analysis with parameters matching the duration and sampling rate of the input trace.

Finally, in addition to using measured signals as inputs to the simulation, the designer can conversely capture traces from the simulated analysis with the "`recordsim`" command, and can inject these recorded signals back into the physical circuit via the AWG. An additional step is necessary when converting a simulated transient analysis into a format that can be programmed into the AWG: Because the transient analysis flexibly varies the length of each time step to increase resolution as necessary, Simpoint resamples this simulated output at a constant sampling rate in order to build a 1024-point lookup table to construct the generated waveform.

## 4.3 Testing Software

Simpoint expands on Strasnick et al.'s software interface [23], and as such is capable of all the debugging techniques described within, such as probing, isolating, recording, and replaying signals. We describe its added capabilities below:

Designers can use the "`modify`" command to manually change the value of a passive component in the physical circuit by rerouting elements from the external component bank in its place. The command takes a target value as an argument and will select the nearest-valued component in the bank. Alternatively, the designer can specify "`up`" or "`down`" to incrementally adjust values to the next highest/lowest available value. With the "`route`" command, Simpoint can also manually route an arbitrary element in the component bank between any two specified points in the circuit, e.g. to compare responses using different ICs.

The "`plotresponses`" command allows the designer to juxtapose measured and simulated responses. It probes signals from each instrumented point on the physical circuit, runs the simulation given the specified input, and produces an interactive visualization of all results.

Single parameter-response analyses can be run via "`plotrelationship`" or "`testrelationship`". The former produces a plot showing changes in the target signal property with respect to the given parameter, while the latter summarizes the relationship as either directly related, inversely related, unrelated, or indeterminate. Those designations are produced as follows:

> We first determine if a can be considered "directly related" to b (which we write as "a $\propto$ b"). a $\propto$ b denotes a monotonic relationship between a and b, which is not necessarily proportional — specifically, it is true when a does not decrease (exceeding tolerance) for any increase in b:
>
> $$(a \propto b) \implies \forall \{j, i \mid b_j > b_i\} : a_j >= a_i \qquad (??.1)$$
>
> We then similarly determine if a is inversely related to b ("a $-\propto$ b") — that is, a does not *increase* for any increase in b. We then check if a is both directly and inversely related to b. This case implies that a neither increases nor decreases for any change in b. We refer to this as a case where a is unrelated to b ("a $!\propto$ b"). That is:
>
> $$((a \propto b) \wedge (a -\propto b)) \Leftrightarrow (a !\propto b) \qquad (??.2)$$

If a $!\propto$ b, we designate that a is unrelated to b
Else if a $\propto$ b, we designate that a is directly related to b
Else if a $-\propto$ b, we designate that a is inversely related to b
Else, we designate that a and b have an indeterminate (non-monotonic) relationship

Both "`plotrelationship`" and "`testrelationship`" run both on the observed hardware and the simulation, allowing for juxtaposition. Before any parameter can be manipulated, the designer must first specify its allowable range using the "`range`" command. Typically, this is the range of interest over which to search for the optimal response, or alternatively, a range bounded by maximum ratings of voltage, current, etc. to ensure that no possible parameter setting would exceed operating limits. Then, to produce these analyses, Simpoint samples several points from the specified parameter range, sets the parameter to each target value, records the response of interest, and calculates any statistics from the captured trace (e.g. mean voltage).

Simpoint then builds upon the individual parameter-response analysis to offer high-level, circuit-wide analyses. The "`whataffects`" command takes as input a target signal property (e.g. mean voltage of signal *s*), and then runs a parameter-response analysis on the target signal for every parameter which has been given an allowable range. The summary reports the nature of each tested parameter's relationship to the target signal (using the same categorization as in "`testrelationship`" above). "`whatdependson`" performs the inverse operation: given a target parameter to manipulate and a signal property of interest (e.g. frequency of the signal at node *X*), Simpoint performs a parameter-response analysis for all possible signals, reporting the nature of relationships to the modified parameter. Both "`whataffects`" and "`whatdependson`" report simulated results in addition to results from the hardware, helping the designer to contrast behaviors.

Finally, Simpoint allows the designer to automate the aforementioned analyses for use in functional testing. Using the "`relation`" command, the designer articulates an expected relationship via the specification language described in Appendix A. When run as a test, parameters in the specification can be either *passively* observed or *actively* manipulated. Consider an example where a designer specifies the frequency of their three-stage RC oscillator as $F_{osc} = \frac{1}{2\pi RC\sqrt{6}}$. If the designer does not specify any parameter ranges for modification, Simpoint will passively check the equality using the current values of $R$ and $C$, akin to a conventional functional test. If instead the designer provides ranges for $R$ and $C$, Simpoint will repeatedly test the relation at sampled values of $R$ and $C$, reporting whether it holds true across the given parameter ranges. Relations that make use of the "increases with" ($\propto$) or "decreases with" ($-\propto$) operators require that the designer provide a range to manipulate at least one parameter. The designer can also configure the tolerance with which two measurements are considered "equal" using either absolute or relative thresholds. Finally, the "`autorelation`" command generates pre-defined specifications to check across the whole circuit, based on common error patterns. In our current prototype, this method evaluates for power integrity issues by checking for any parameters with a significant relationship to the peak-to-peak voltage or mean voltage of the supply rails.

### 4.4 Evaluated Technical Metrics

Like all forms of instrumentation, interfacing a PCB with Simpoint adds parasitic factors that should be considered for sensitive circuits. Each instrumentation point in Simpoint adds .8Ω of on-state resistance into the trace (measured tip-to-tip across the jumper pad), with a capacitance of .17 nF measured between the ends of each jumper pad in the off state. A programmable connection across a row of the component bank incurs an additional 16 Ω of resistance (attributed to the $R_{on}$ of the analog multiplexers), with a parallel capacitance of .41 nF. The designer can maximize accuracy of parameter-response analyses by considering these factors when specifying the value of components in the component bank. Each channel of the USB oscilloscope and AWG has an impedance of 1MΩ / 10pF. We optionally attach a unity gain buffer to the AWG to allow injected signals to source greater current. Jointly considering the overall instrumentation load, Simpoint performs best in low-frequency and low-current applications, whereas designers may want to avoid instrumenting high-frequency signals such as oscillators.

The exact time to complete each analysis varies depending on the dynamics of the circuit and the simulation. By default, after each hardware modification, the system waits a configurable amount of time before taking measurements, to avoid capturing transients and to ensure that the circuit settles into a new steady-state. The designer can set this minimum delay according to the expected rise and fall times within their application. Examples in this paper used the default 1 ms delay. The runtime for each simulation can also vary considerably depending upon how quickly values converge. In the example applications in this paper, Simpoint completed each individual analysis (perform a modification, capture the measurement, and run the corresponding simulation) in roughly .5 seconds on average. Thus, a parameter-response analysis sampling 10 parameter values would run in several seconds.

## 5 DEMONSTRATIONS IN CONTEXT

In this section, we evaluate Simpoint's utility by demonstrating its usage on a range of example debugging scenarios. We begin with a deep-dive on a single circuit, showing how each of Simpoint's core operations can prove helpful for different classes of bugs. We then show a breadth of other circuits and techniques that leverage Simpoint's capabilities in additional ways.

Examples A1–A3 relate to the circuit shown in Figure 5, known as a Single-Ended-to-Differential converter. Given an input voltage, it produces a differential output pair, a format often used in transmission due to increased noise rejection. The converter in Figure 5 takes in a single-ended input ranging from .1V to 3.2V and linearly maps it to a differential output ranging from -3.1V to 3.1V.

### 5.1 Example A1: Comparing Measured and Simulated Responses to Localize a Fabrication Error

Bugs often occur when errors in fabrication or assembly result in a mismatch between the intended design and the constructed hardware. The foremost challenge in these cases is to localize where observed behaviors deviate from expectations (or first deviate, in a chain of inputs and outputs). In this example, the output pin

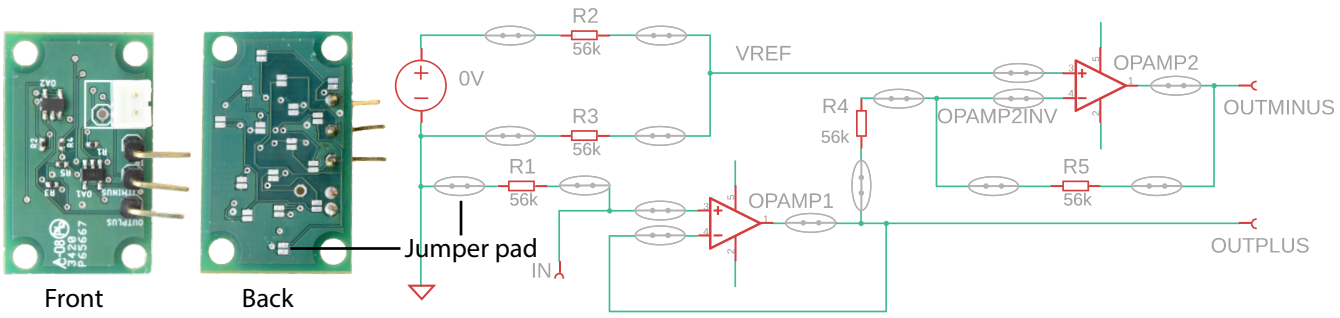


**Figure 5: An example circuit which converts a single-ended input signal ("*in*") ranging from .1V to 3.2V into a differential signal ("*outplus*" and "*outminus*") ranging from -3.1V to 3.1V.**

of the *OPAMP2* component has been damaged during or prior to fabrication, resulting in a poor connection to the board (Figure 6.A). As a result, the measured output (*outminus*) does not actually reflect an amplifier in feedback, but rather is solely driven by the signal at the inverting terminal (*opamp2inv*). This type of fault is challenging to detect visually or via continuity testing, as the exposed portion of the pin may still appear well connected.

Figure 6 describes how we use the "`plotresponses`" command to compare measured signals with their simulated counterparts across the signal chain to find the start of error behaviors. Then, to localize where the fault lies along the amplifier's feedback path, we programmatically modify the circuit and compare how the responses change both on the hardware and simulation. Using these tools, we find that a simulation of the circuit in which the output of the second op-amp is disconnected perfectly matches the behavior of the hardware, and therefore conclude that the fault likely lies at this location.

## 5.2 Example A2: Analyzing Parameter-Response Relationships to Characterize Error Behaviors

Debugging errors frequently requires gathering additional observations to understand patterns of behavior. The application in this example calls for using the Single-Ended-to-Differential Converter to convert a .1V input into a -3.1V differential output ($V_{outplus} = .1V, V_{outminus} = 3.2V$). Instead, we observe a value of 2.1 V for $V_{outminus}$ when the .1V input is connected, resulting in a -2.0V differential output. There are several issues that could result in this behavior, and so we aim to characterize the pattern of behavior beyond the one data point: Is the output incorrect for all inputs, or just some subset? Do the outputs differ from the specification by a constant offset or by a scaling factor, or is the output simply stuck at 2.1V?

To characterize the patterns of behavior in our circuit, we first use the "`plotrelationship`" command to generate a plot as in Figure 7.A,C,E that compares the simulated and measured responses at *outminus* as the input voltage is varied across its range. Figure 7.A shows a reference for what we expect to see on a bug-free version of the circuit, whereas our observed output is shown in Figure 7.C. The difference in the slopes of the input-output relationship on hardware implies that our output differs by a scaling factor. Given that the gain of the amplifier is controlled by a resistor ratio, we suspect that the difference is caused by excess resistance in some component or on the board itself. Then, to determine how to adjust the resistors in our circuit to achieve the correct response, we use a second "`plotrelationship`" command to visualize how the response to the .1V input varies as a function of the feedback resistor R5 (Figures 7.B,D,F). From Figure 7.D, we can read off the value of R5 which produces the correct output.

To highlight how characterizing the pattern of behavior aids in our reasoning about the cause of the error, Figures 7.E-F depict a different bug that would have produced the same output at our .1V input. Instead of reduced gain, these relationships depict early saturation of the output, implying that the op-amp does not have a wide enough dynamic range for this application. Solving this bug would instead require us to increase the supply voltage or use a different (rail-to-rail) op-amp. While these two bugs present similarly in the initial application context, analyzing their full parameter ranges reveals distinct patterns of behavior.

## 5.3 Example A3: Scanning Relationships Across the Circuit to Detect Couplings

Real-world circuits are also subject to various forms of couplings (e.g. noise) that often are not modeled in the simulation. Couplings manifest as unexpected dependencies within the circuit, and debugging involves identifying and characterizing them. In this example, we find that the Single-Ended-to-Differential Converter does not yield correct outputs for AC inputs. Comparing observed and simulated responses shows that the reference voltage ($V_{ref}$) does not remain constant on the hardware, and instead is affected by the frequency of the input (Figure 8.A). We hypothesize that there may be capacitive coupling between traces on the board, which could occur anywhere along the signal chain. Localizing the site of this coupling would typically involve isolating and testing all possible pairs of points in the circuit. Instead, we make use of circuit-wide analysis to detect the unintended coupling.

We begin by opening all jumper pads in the circuit as shown in Figure 8.B, which disconnects all nodes both on the physical circuit and in the coupled simulation. As a result, a test signal applied at one point will only propagate if some other coupling exists. We then use the "`whataffects`" command to query for all signals

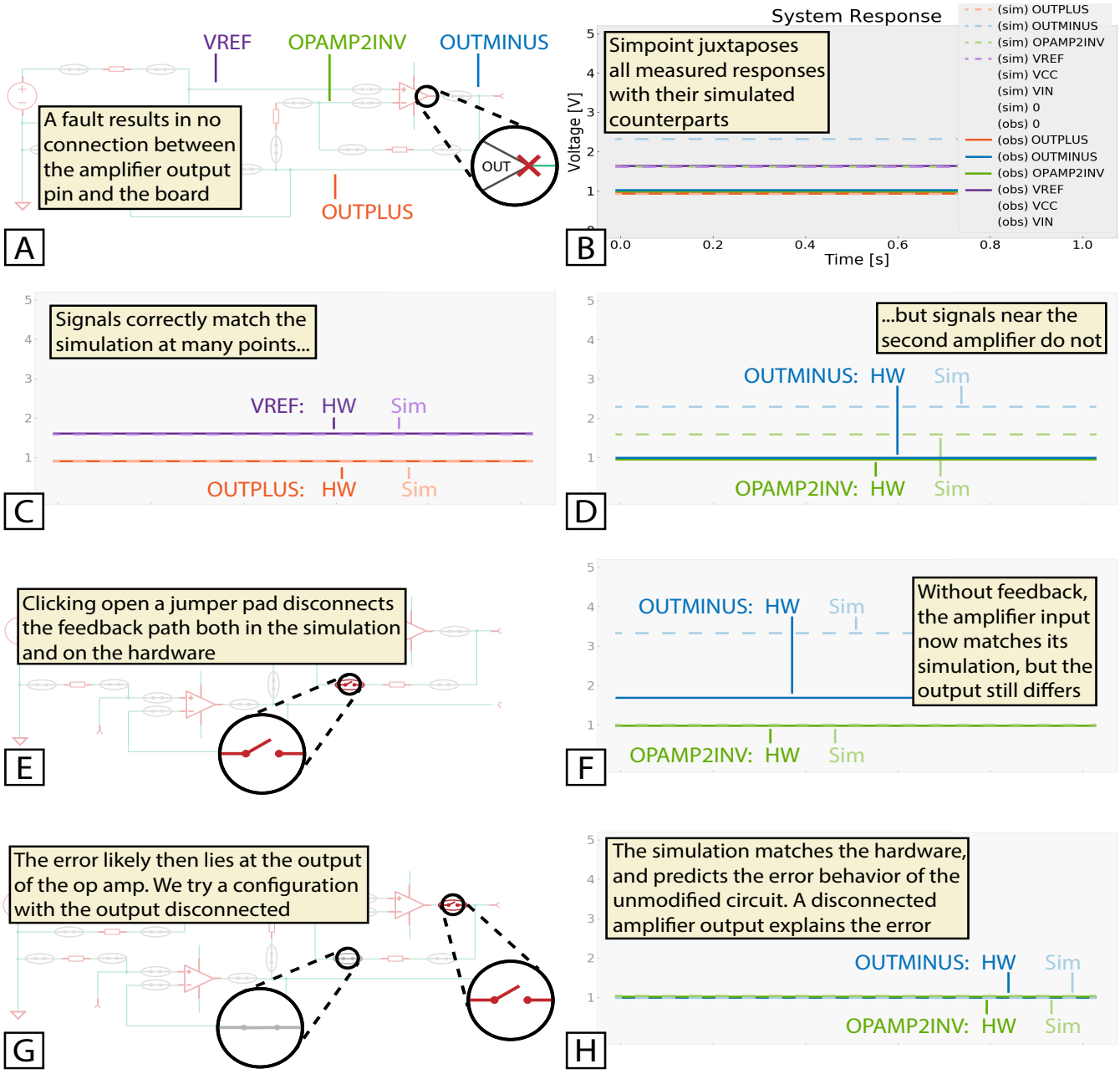


**Figure 6: Debugging a fabrication error on the Single-Ended-to-Differential converter in Figure 5. A fault (A) results in no connection to the output pin of the second op-amp. Observing that the differential output from the hardware is always zero, we first use the "`plotresponses`" command to compare all responses in the circuit with their simulated counterparts (B). Highlighting signals along the chain from input to output, we find that the first amplifier stage and reference voltage match the simulation (C), while deviations begin at the second amplifier (D). However, as these two signals are in feedback, it can be difficult to discern whether the error occurs at the op-amp's input or output, internal to the op-amp, or along the feedback path. Guessing first that the issue lies along the feedback path, we toggle a jumper pad to temporarily interrupt the feedback on both the physical and simulated circuits (E). We compare signals again following this modification (F), which shows that the input to the amplifier matches its simulation, but the output still differs. Now suspecting that the output is the issue, we restore the feedback connection and instead toggle a pad to disconnect the output pin from the circuit (G). In this configuration, the simulated behaviors perfectly match the behaviors of the hardware (H), demonstrating that a disconnected output pin would explain the original bug.**

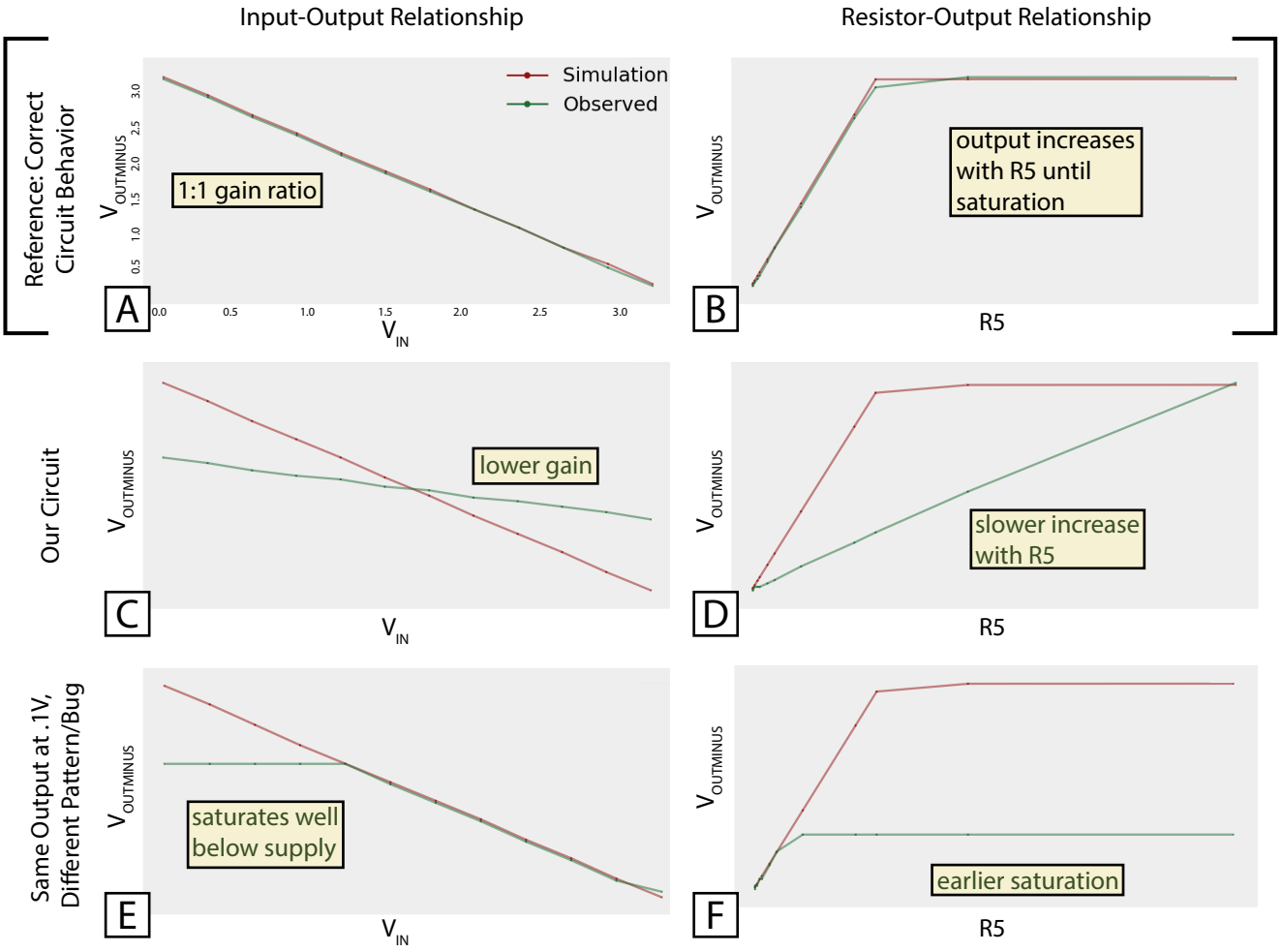


**Figure 7: Using the "`plotrelationship`" command to analyze and compare parameter-response relationships both on the physical circuit and in simulation. A) The correct relationship between input ($V_{in}$) and output ($V_{outminus}$) signals. B) The correct relationship between $V_{outminus}$ and the value of resistor R5, for a .1V input. C-D) The same relationships on our circuit, showing a reduced gain ratio leading to incorrect output. The plot in (D) can be used to determine a new resistor value that will produce the correct output. E-F) The same relationships for a hypothetical bug that would have result in the same erroneous output for the .1V input. Here the op-amp component has a limited dynamic range below the supply rail (measured on a version of the circuit with the LM324 instead of the TLV2371).**

whose frequencies still affect the frequency observed at $vref$. The summary reports that the frequency observed at $vref$ varies with the frequency of signals applied to *outplus* on the hardware, while no such relationship exists in the simulation. Thus, the designer concludes that the coupling occurs between *outplus* and $vref$. To ensure that we detect any further couplings that may appear in future testing, we save behavioral specifications for each signal, stating that none should have a significant frequency-frequency relationship with $vref$.

## 5.4 Example B: Manual Component Tuning to Calibrate a Current Sensor

In addition to automatically modifying components as part of a parameter sweep, Simpoint makes it easy for the designer to manually adjust component values to tune a response through software control. We designed a low-side current sensor based on AN3222 from STMicroelectronics [22], shown in Figure 9. It amplifies the voltage differential across a small-value shunt resistor, as determined by ratios of the surrounding feedback resistors. Assuming ideal amplifier characteristics, we specify the output in terms of the shunted current as

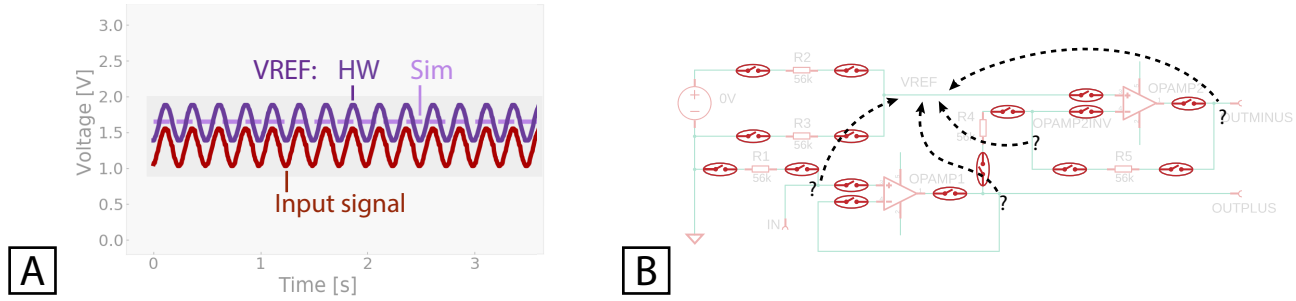


**Figure 8: Locating noise couplings using circuit-wide analysis commands. A) We note that the reference voltage ($V_{vref}$) varies with AC inputs, possibly caused by capacitive coupling somewhere in the circuit. B) To find which node is being coupled to $vref$, we first disconnect all nodes from each other (in both the simulation and the physical circuit) by clicking opens all jumper pads. Then, we use the "`whataffects`" command to query for circuit nodes where the frequency of applied signals still causes changes to the frequency of signals observed at $vref$. The summary reports a relationship at the $outplus$ node, suggesting that the coupling occurs between $outplus$ and $vref$.**

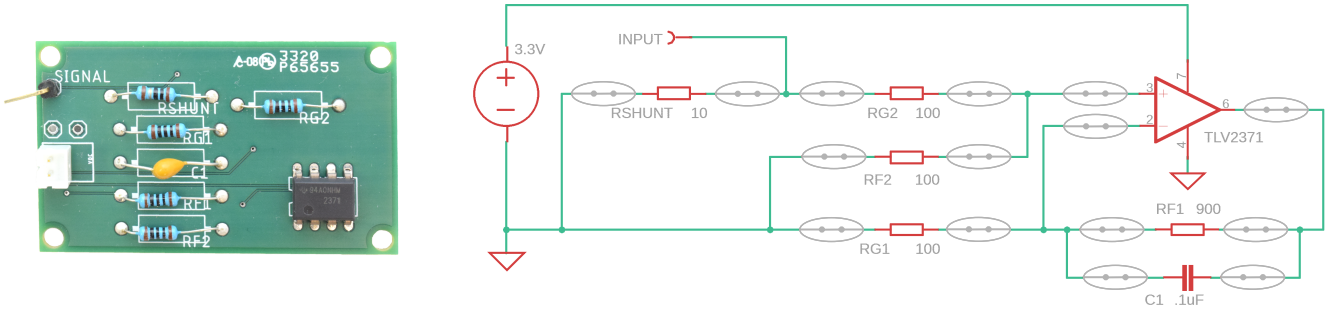


**Figure 9: A low-side current sensor, whose resistor ratios can be tuned to amplify the output signal.**

$$V_{out} = I * R_{shunt}(1 - \frac{R_{G2}}{R_{F2} + R_{G2}})(1 + \frac{R_{F1}}{R_{G1}}) \quad \text{(B.1)}$$

However, this relationship omits error terms caused by non-idealities of the physical circuit (e.g. leakage current at the op-amp inputs). For this reason, once fabricated the circuit may need tuning of values to optimize the response. We can compare simulated and measured outputs to quantify the amount of error for different test inputs. Then, rather than manually replacing each resistor and repeating tests, Simpoint allows us to programmatically modify each resistor up or down across the set of values provided in the component bank, or similarly to plot the response of interest with respect to resistances across the range. These operations reduce the time necessary to empirically validate the optimal resistor values.

## 5.5 Example C: Detecting Supply Noise in a Temperature-Controlled Fan via Automatically-Generated Checks

Simpoint can automatically generate specifications to flag relationships that suggest error in most circuits. For example, any parameter found to influence the supply voltage likely suggests a power integrity issue, such as ripples or voltage sags. Simpoint flags one such instance in the circuit in Figure 10, which uses an analog temperature sensor (TMP36 [2]) along with a comparator and transistor to switch a DC fan when the surrounding temperature rises above a set threshold. When we begin debugging to identify why the activation threshold seems to drift during operation, the "`autorelation`" command detects unexpected relationships on the hardware between the supply voltage $VCC$ and several parameters, including the sensor output and comparator output, as well as the fan load resistance. From these unintended relationships, we can quickly deduce that switching the load is causing noise on our power supply lines, requiring additional decoupling.

## 5.6 Example D: Debugging the Simulation to Identify Design Issues in a Clap Detector

We also acknowledge the many occasions in which designers fabricate circuits without first validating the design in simulation. In this application we discuss such an example, and how the simulation generated automatically by Simpoint during instrumentation can aid designers in debugging design flaws.

We replicated the 29132 Sound Impact Sensor for Parallax, Inc. [18], shown in Figure 11. It uses an electret condensor microphone, whose signal is AC coupled and amplified by a common-emitter transistor amplifier, before being used to trigger a 555 timer IC to drive an LED output. When the fabricated sensor fails to detect inputs, we connect Simpoint for debugging and specify expected behaviors. For example, the gain of the common-emitter amplifier

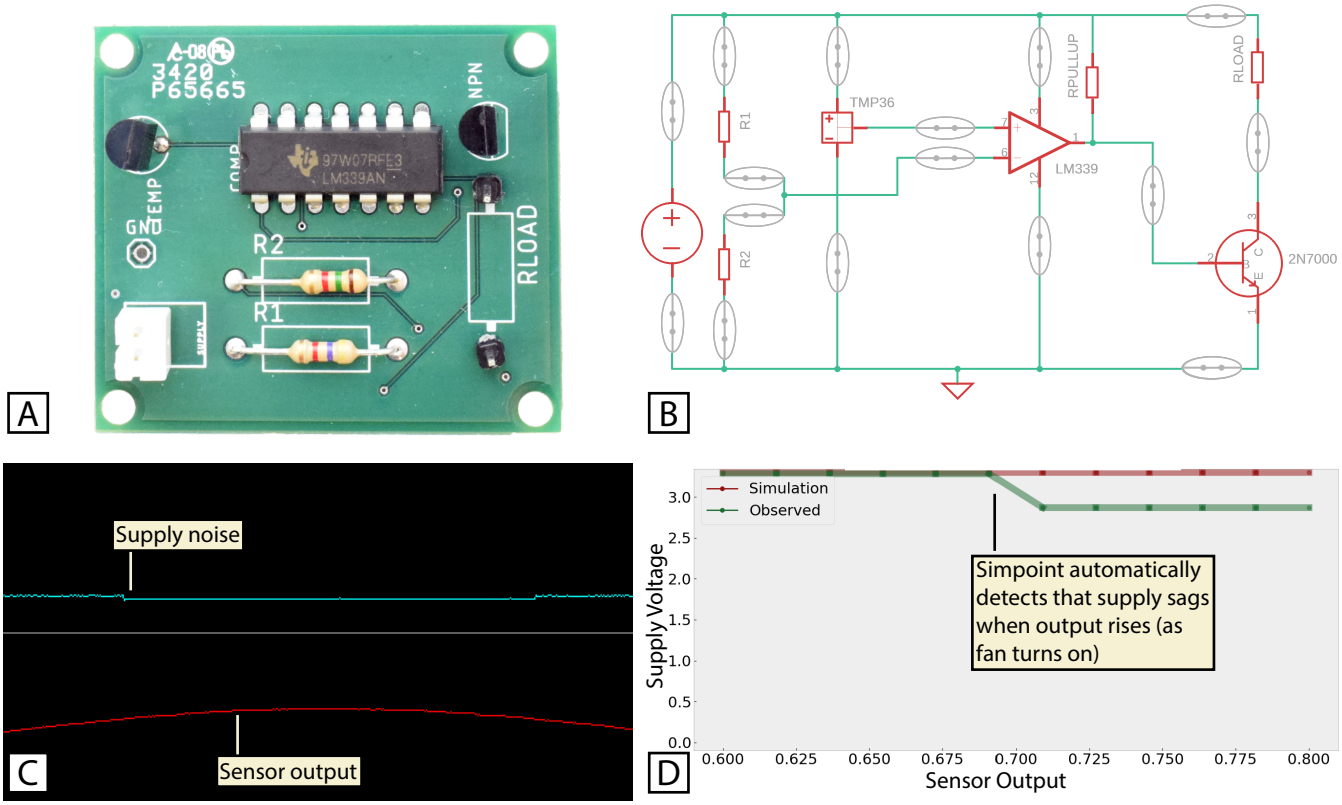


**Figure 10: A fan (represented by $R_{load}$) controlled by a temperature sensor. Switching the fan causes small changes in the supply voltage (C). Simpoint can automatically detect and flag this relationship (D) as a likely source of error.**

should be proportional to the ratio of the collector and emitter resistors – expressed here using the peak-to-peak voltage of the AC signal:

$$Vpp_{collector} \propto \frac{R_3}{R_5} \tag{D.1}$$

Simpoint notifies us that this specification fails both on the observed hardware *and in simulation*, which implies a problem with our design rather than a fabrication error. To tease out why the amplifier isn't working, we investigate lower-level assumptions about how the amplifier works. The operating point of the transistor base differs widely between the simulated and observed versions. Which version is correct? In the "self-biased" configuration of our circuit, the bias point should be set by current flowing across the collector-feedback resistor R2, with larger values of R2 causing a larger voltage drop. We can specify:

$$Vavg_{base} < Vavg_{collector} \tag{D.2}$$

$$Vavg_{base} \; -\propto \; R2 \tag{D.3}$$

However, in both our simulated and observed circuits, the operating point of *base* and *collector* are equal (Figure 11.C), causing specification D.2 to fail. Similarly, a parameter-response analysis reveals that neither the measured nor simulated operating points of the transistor base varies with R2 (Figure 11.D), causing a failure of specification D.3. These failures suggest that current isn't flowing into the base junction at all, despite current across the collector-emitter junction. This behavior highlights the error in our design: our implementation used a MOSFET, instead of the BJT depicted in the Parallax implementation. Without current flowing over R2, the collector-feedback configuration does not properly set the bias point. Switching to a BJT or using an alternate biasing approach solves the issue. Notably, had we not examined behaviors of the simulation debugging, we might have continued to assume that our failure to replicate the Parallax implementation was due to a defect in the physical circuit, rather than considering flaws in our design choices.

## 6 LIMITATIONS AND FUTURE WORK

Below we describe future opportunities to improve Simpoint's design, as well as fundamental limitations of the approach:

In addition to the finite size of its component bank, the current implementation of Simpoint only allows for the substitution of a single, two-terminal component at a time (separate from varying parameters of an injected signal). While the use of an analog crosspoint array would allow for an arbitrary number of connections between instrumentation sites and banked components, commercially available crosspoint devices have significantly higher parasitic impedance than our multiplexer-based approach, which would limit usage of the tool for many classes of circuits. We also point out that Simpoint gathers sets of measurements in series (such as when

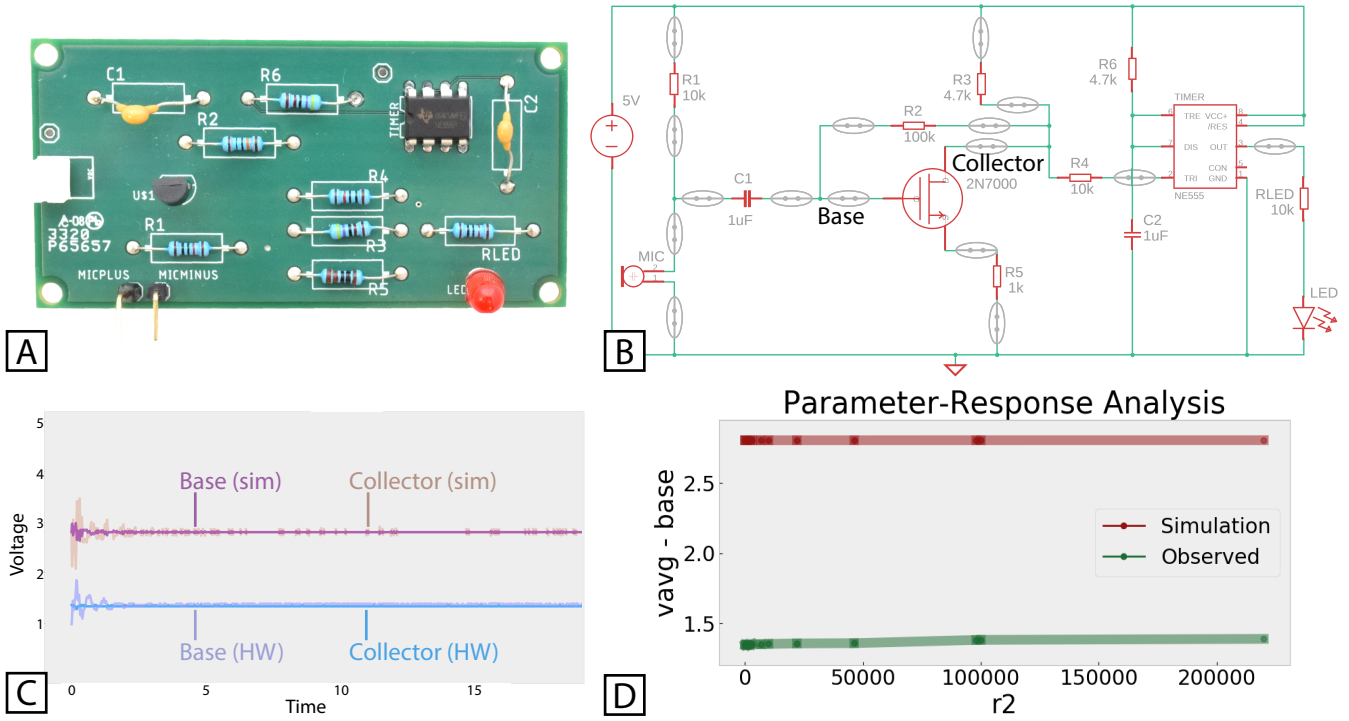


**Figure 11: A-B) A "clapper" circuit which lights an LED after detecting a loud noise. C) Using the "`plotresponses`" command to determine that neither the measured nor simulated signals show an expected voltage drop across R2. D) Using the "`plotrelationship`" command to determine that the operating point of the transistor base does not vary with the value of R2. These findings lead us to revisit the design and question why our chosen transistor cannot be used in a collector-feedback configuration as shown.**

scanning responses across the circuit), rather than simultaneously. Thus, for examining the response to transient events, the designer should make use of the built-in record and replay features to capture and loop the event during analysis. Adding more oscilloscope channels would allow for true parallel inspection.

As Simpoint cannot truly "modify" discrete on-board components but rather replaces them, it is possible for unexpected differences to arise when modifying values, e.g. if one of the banked components is itself faulty. Designers should take care to match the types (e.g. manufacturers) of components in the component bank to those on their board, so as to minimize differences beyond the manipulated parameter.

Currently, Simpoint supports analog simulation only, although mixed-signal variants of SPICE and other simulation languages have been developed. Future iterations of Simpoint can couple mixed-signal simulation alongside software debugging techniques to better support the debugging of embedded systems.

As with the Pinpoint system [23], the jig interface generated by Simpoint is itself a point of potential failure in the debugging environment. In theory, bugs resulting from faults in the jig construction would be challenging to diagnose through the use of Simpoint alone. Currently, the jig itself is not explicitly modeled within the simulation (as the parasitic factors we measured on the jigs were orders of magnitude smaller than those introduced by the relays and multiplexers).

Finally, we acknowledge existing limitations in the accuracy of simulation that prevent it from being an accurate benchmark for some applications. A simulation is only as accurate as the models it is given, and thus imperfect component models or omitted factors (such as noise or temperature variation) can cause "true" theoretical behaviors to differ somewhat from simulated output. Further, there are many components which are fundamentally challenging to represent in a SPICE model, such as electromechanical devices whose electrical properties depend upon mechanical dynamics. Simpoint therefore is not intended to absolve the designer of their responsibility to consider the expected behaviors of their circuit, as the simulation itself is not foolproof. As one means of improving confidence in the simulation, by articulating expected behaviors through Simpoint's specification language, the designer can be alerted when the simulation itself fails a given test.

Moving forward, we envision further applications for coupling simulation with live debugging to assist designers. For example, similar capabilities for programmatically modifying hardware parameters would enable us to automate fault modeling approaches on hardware. By automatically injecting faults in both hardware and simulation and compiling the resulting responses, tools can present designers with candidate faults which are likely to match

encountered errors. Learning-based systems can further automate the generation of debugging hypotheses by training on labeled (or fault-injected) patterns of error manifestation and classifying behaviors on the designer's circuit.

Additionally, while this paper explores how Simpoint's techniques provide utility in real application scenarios, we also consider broader questions about how they might integrate into existing workflows and fulfill different functions depending on the expertise and application domains of its users. The next step for this work therefore calls for long-term deployment studies to further examine usages across different contexts.

## 7 CONCLUSION

Simpoint explores the utility of tightly coupling simulated and physical versions of the same circuit to support interactive debugging. By updating the simulation in response to hardware modifications, it provides an always-available juxtaposition with expected responses to the designer's inputs. By augmenting a standard PCB to enable programmatic component modifications, it allows for shared parameter-response analyses across both simulation and hardware. It is our goal that by reducing barriers to the use of simulation in debugging, we can increase designers' access to a broader range of debugging techniques.

## REFERENCES


[1] [n.d.]. Ngspice – The open source spice circuit simulator. http://ngspice.sourceforge.net/

[2] Analog Devices. 2015. TMP35/TMP36/TMP37. https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35{_}36{_}37.pdf

[3] Sameh Attia and Vaughn Betz. 2020. StateMover: Combining Simulation and Hardware Execution for Efficient FPGA Debugging. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '20)*. Association for Computing Machinery, New York, NY, USA, 175–185. https://doi.org/10.1145/3373087.3375307

[4] Autodesk. 2018. EAGLE PCB Design Software. https://www.autodesk.com/products/eagle/overview

[5] M Bacic. 2005. On hardware-in-the-loop simulation. In *Proceedings of the 44th IEEE Conference on Decision and Control*. 3194–3198. https://doi.org/10.1109/CDC.2005.1582653

[6] S Banerjee and T Gupta. 2012. Fast and scalable hybrid functional verification and debug with dynamically reconfigurable co-simulation. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 115–122.

[7] P Caunegre and C Abraham. 1995. Achieving simulation-based test program verification and fault simulation capabilities for mixed-signal systems. In *Proceedings the European Design and Test Conference. ED TC 1995*. 469–477. https://doi.org/10.1109/EDTC.1995.470318

[8] Analog Devices. 2018. ADG732 Datasheet and Product Info. http://www.analog.com/en/products/adg732.html

[9] Pragun Goyal, Harshit Agrawal, Joseph A. Paradiso, and Pattie Maes. 2013. BoardLab. In *Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology - UIST '13 Adjunct*. ACM Press, New York, New York, USA, 19–20. https://doi.org/10.1145/2508468.2514936

[10] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design as Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/1449715.1449732

[11] Lipeng Ji and Xiaohui Hu. 2018. Analog circuit soft-fault diagnosis based on sensitivity analysis with minimum fault number rule. *Analog Integrated Circuits and Signal Processing* 95, 1 (apr 2018), 163–171. https://doi.org/10.1007/s10470-018-1111-y

[12] JTAG Technologies. 2018. Boundary Scan In-Circuit Programming. https://www.jtag.com/

[13] Mitchell Karchemsky, J.D. Zamfirescu-Pereira, Kuan-Ju Wu, François Guimbretière, and Björn Hartmann. 2019. Heimdall: A Remotely Controlled Inspection Workbench For Debugging Microcontroller Projects. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300728

[14] Yoonji Kim, Youngkyung Choi, Hyein Lee, Geehyuk Lee, and Andrea Bianchi. 2019. VirtualComponent: A Mixed-Reality Tool for Designing and Tuning Breadboarded Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300407

[15] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 389–400. https://doi.org/10.1145/258734.258887

[16] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and Checking Behavior of Embedded Systems Across Hardware and Software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17*. ACM Press, New York, NY, USA, 299–310. https://doi.org/10.1145/3126594.3126658

[17] Laurence W. Nagel and D.O. Pederson. 1973. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Technical Report UCB/ERL M382. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[18] Parallax Inc. 2020. Sound Impact Sensor | 29132. https://www.parallax.com/product/29132

[19] PJRC. 2020. Teensy USB Development Board. https://www.pjrc.com/store/teensy36.html

[20] Fabrice Salvaire. [n.d.]. PySpice: Simulate electronic circuit using Python and the Ngspice / Xyce simulators. https://github.com/FabriceSalvaire/PySpice

[21] Mustapha Siamani and Bozena Kaminsi. 1992. Analog Circuit Fault Diagnosis Based on Sensitivity Computation and Functional Testing. *IEEE Design and Test of Computers* 9, 1 (1992), 30–39. https://doi.org/10.1109/54.124515

[22] STMicroelectronics. 2010. AN3222 Application Note. https://www.st.com/content/ccc/resource/technical/document/application{_}note/d5/2e/49/2b/29/73/4c/46/CD00274901.pdf/files/CD00274901.pdf/jcr:content/translations/en.CD00274901.pdf

[23] Evan Strasnick, Sean Follmer, and Maneesh Agrawala. 2019. Pinpoint: A PCB Debugging Pipeline Using Interruptible Routing and Instrumentation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. New York, New York, USA, 1–11.

[24] Coto Technology. 2020. CotoMOS® C226S/C326S. https://www.cotorelay.com/product/c226s{_}c326s{_}series{_}mosfetrelay/

[25] Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 711–718. https://doi.org/10.1145/985692.985782

[26] K A Tomko and A Tiwari. 2000. Hardware/software co-debugging for reconfigurable computing. In *Proceedings IEEE International High-Level Design Validation and Test Workshop (Cat. No.PR00786)*. 59–63. https://doi.org/10.1109/HLDVT.2000.889560

[27] Te-Yen Wu, Mike Y. Chen, Bryan Wang, Jiun-Yu Lee, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, and Yu-Chih Lin. 2017. CircuitSense. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17*. ACM Press, New York, New York, USA, 311–319. https://doi.org/10.1145/3126594.3126634

[28] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. 2017. CurrentViz. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17*. ACM Press, New York, New York, USA, 343–349. https://doi.org/10.1145/3126594.3126646


## A SPECIFICATION LANGUAGE

We describe Simpoint's language for articulating behavioral specifications, with terminals in CAPITALS and nonterminals in `teletype` font. All top-level specifications are a `statement`, consisting of one or more `relations` and logical operators. Valid `statements` include:

- `relation`
- IF `statement` THEN `statement` (ELSE `statement`)
- `statement` AND `statement`
- `statement` OR `statement`

- NOT `statement`

Each `relation` uses a single binary operator. These operators include numerical comparisons (=, <, >, <=, >=), which evaluate the operands using the provided tolerance value, and relationship operators ($\propto$, $-\propto$, $!\propto$).

"a $\propto$ b" denotes a monotonic relationship between a and b, which is not necessarily proportional – specifically, it is true when a does not decrease (exceeding tolerance) for any increase in b:

$$(a \propto b) \implies \forall\{j, i \mid b_j > b_i\} : a_j >= a_i \tag{A.1}$$

Similarly, a $-\propto$ b is true when a does not increase for any increase in b. a $!\propto$ b denotes that a neither increases nor decreases for any increase in b. We therefore note that:

$$((a \propto b) \wedge (a -\propto b)) \Leftrightarrow (a !\propto b) \tag{A.2}$$

For convenience, "$\sim$" may be used in place of $\propto$ to type `relations`.

The operands of each `relation` can be a `componentparameter`, a `signalproperty`, an `expression`, or a numeric literal. `componentparameters` include resistance, denoted R.<component name>, and capacitance, denoted C.<component name>. `signalpropertys` include:

- V.<signal name> (instantaneous voltage)
- VAVG.<signal name> (average voltage)
- VPP.<signal name> (peak-to-peak voltage)
- VMIN.<signal name> (minimum voltage)
- VMAX.<signal name> (maximum voltage)
- F.<signal name> (frequency)
- P.<signal name> (period)

`expressions` allow for mathematical computations on measurements. They consist of a numerical operator (+, −, ∗, /, ^) and two operands, each of which can be a `componentparameter`, a `signalproperty`, an `expression`, or a numeric literal.

Finally, parens can be used in specifications resolve ambiguities through grouping.

Once measurements are taken, specifications are evaluated as follows:

- `parameters` are replaced by their value at the time of measurement
- `signalpropertys` are calculated based on the measured (or simulated) signals
- `expressions` are evaluated numerically
- `relations` are evaluated as true / false
- `statements` are evaluated as true / false