Self-Diagnosis Of Electronics With Analog Circuits
Introductory Concepts
by Dennis L Feucht

The trend in maintenance of electronic consumer devices has been to avoid the problem. If it breaks, throw it away and buy a new one. While this approach has not made major in-roads into industrial equipment such as test and measurement (T&M) or medical instruments or factory automation, it does suggest that renewed attention might be given the problem of how to diagnose and repair faulty units once they are in the field. With cheap computing nowadays, perhaps it is time to revisit the use of embedded artificial intelligence (AI) applied to diagnostics.

As electronic equipment has become more complicated, component-level repair at the customer site has become awkward, like trying to fix a car under a shade tree instead of in a properly-equipped garage. Consequently, companies have resorted to the policy of board-level replacement at the point of use. The replaced board is brought back to a service center where it is repaired on a test bench. As boards have become surface-mount and more complicated, they are returned to the factory where an elaborate test center fixes them.

Returning a cell phone or calculator to the factory for repair is not cost-effective relative to purchase of a new unit. The result is that the garbage dumps are strewn with otherwise valuable electronics for lack of knowledge of which low-cost part to replace. (The same problem exists for automobiles. A leaking water pump might have a bad seal, but the entire pump assembly is replaced instead because it is not cost-effective for the garage to tear the pump apart to replace it.) This seems suboptimal and bothers those of us who oppose needless waste.

What is the next step in the evolution of field maintenance? Increasingly, electronic products contain embedded computing, and this computing is not only low in cost relative to the total product but continually increasing in instruction execution rate at comparable or lower power dissipation. Concurrently, the observability of ICs is increasing: digital ICs due in large part to JTAG, and analog ICs (potentially at least) due to the increase in package pin count and the decrease in transistor cost in ICs, allowing test circuitry to be built into them. This circuitry could be as simple as additional buffers from internal nodes, brought out to pins. Or it could include entire monolithic data acquisition systems and an analog form of JTAG, which multiplexes out analog quantities in sync with serial-digital node addressing. Perhaps an extension of JTAG would be the route to take. Companies specializing in on-chip diagnostic subsystems would arise (for instance, Keith Lofstrom does this http://www.kl-ic.com) to augment IC designs.

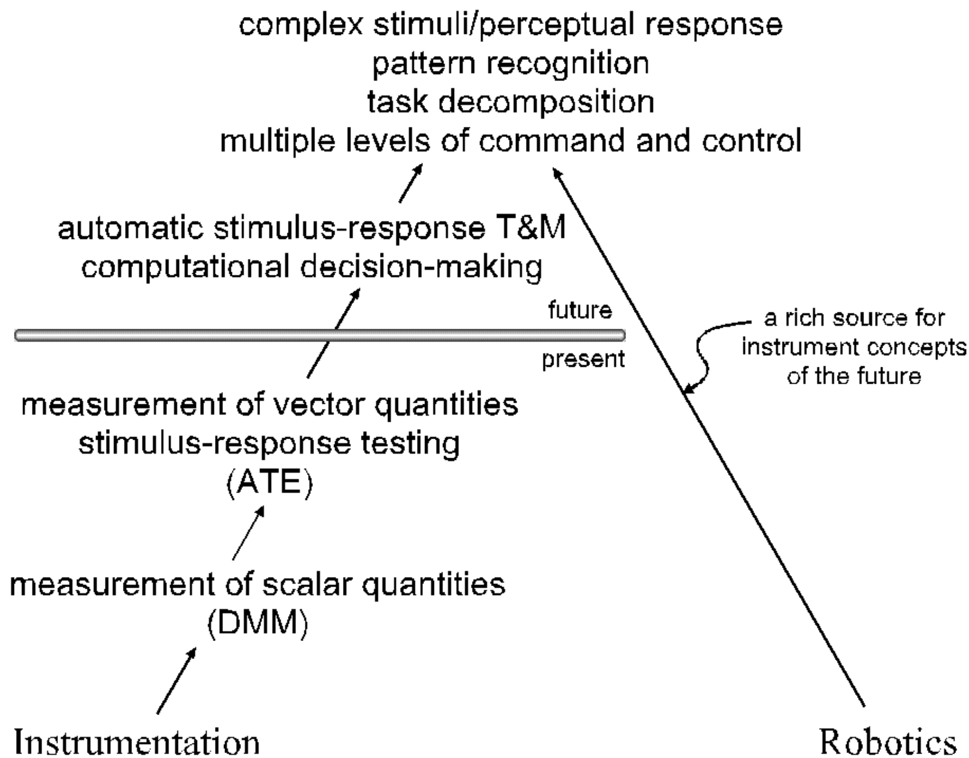
Embedded Designs

Built-in diagnosis can be performed using a microcontroller (µC) already in the system. The cost would be that of additional program ROM and perhaps a slight addition of RAM, plus hardware for increased observability and controllability. Diagnosis is usually modal, not real-time, and can use existing system resources in diagnostic mode. The cost must justify placing a diagnostic system -- the functional equivalent of what is otherwise needed on a test bench, including technician -- in each unit produced. Happily, this is becoming feasible.

Within typical product development nowadays is creation of a test plan, which usually involves a troubleshooting tree that a factory technician follows to find faults in newly-manufactured products. This approach migrated long ago to automated test equipment (ATE) whereby the test

computer runs a diagnostic program (usually one per board), supported by its extensive ATE waveform acquisition and generation hardware. The next step in the progression is to place enough distributed test circuitry into the product itself to support the diagnostics. One of the possible benefits is that the cumbersome task of creating all the "test vectors" for a product can be avoided by giving a general-purpose software diagnostician a high-level definition of the product. From it, built-in knowledge about waveforms, circuits, and systems is applied to the product definition using diagnostic reasoning. Is this feasible?

In the early 1980s, I was working on such an effort in the now-defunct Tek Labs Systems and Cybernetics Group of Tektronix, trying to follow a research path as shown below. In this diagram I consider AI under the preferred heading of *robotics* because the ultimate robot entails all of what constitutes AI plus advancements in hardware: electronic, mechanical, chemical, and combinations thereof. In other words, the embedded *intelligent instrument* is a kind of robot.



The development of electronics diagnosis has begun with the present manually-reasoned diagnostic tree. What would it take to simply automate it? More than might be evident at first. A technician following the tree has an immense knowledgebase which includes reasoning not only about diagnosis itself, but also about test, measurement, waveforms, and circuits. Both observation and reasoning at these multiple levels of abstraction are combined in diagnosis. To reason requires a model of what is being reasoned about. The model *could* consist merely of a table relating faults to observable states (that is, to measurements at a given setting) of the object under test. If the object is sufficiently observable, each fault should correspond to a unique state.

One weakness of this approach is that the resulting diagnosis is empirically intensive, resulting in many measurements. Measurements have a cost and the cost of diagnosis is what we want to reduce through automation. To minimize measurements deeper knowledge is required than is found in a decision tree. A set of diagnostic rules can capture the knowledge of a decision tree. Instead of storing all the nodes of the tree at compile time, they are deduced at run-time from

basic rules. The sequence of deductions enumerates a possible path through the complete tree of all possible paths and the test engineer does not have to make every path explicit.

With this approach, each new product would require a diagnostic rule-set. To reduce this effort, a more general approach can be taken of embedding a knowledge-based diagnostician (KBD) with built-in rules for diagnosis, circuit analysis, waveform analysis, testing, and instrumentation. Then each new product design would only need a specification of the product to be given to the KBD, thereby reducing diagnosis to a relatively easy task.

Functional And Parametric Faults

Before working on a scheme to do diagnosis, we should briefly look at types of faults. Faults can be categorized as shown in the following table.

Faults	**Effect** (symptom of system failure)		
		Parameter	Function
Cause (component failure)	Parameter	predictive & prevented by periodic recalibration	usually due to design faults
	Function	failure of non-critical component	catastrophic failure

A *parametric* fault causes the object to fail to meet performance specifications. A *functional* fault effect is qualitative; the object under test does not perform at all, at least in a given mode (for particular settings) of operation. Usually parametric effects are recognizable as such and are not as serious as functional failures because the object continues to function, though not adequately. (For instance, a car without a cooling fan functions completely, though its operating time is limited before functional faults occur.) The out-of-specification parameter is usually (though not always) readily identifiable and corrected through recalibration. To correct this kind of fault requires the automated diagnostician to have both calibration knowledge and the ability to make adjustments electronically. If not a human is required, who is told which adjustment to tweak. In this case the person need not have the skills of a technician, and it might be possible for the user to actually do the repair by following instructions from the built-in diagnostician. A functional failure leading to a parametric fault effect is like a person without tonsils, spleen, or appendix, who completely functions, but at diminished capacity. The failed circuit does not cause malfunction but, rather, suboptimal function. These faults should be included in diagnosis.

The more serious faults are functional (or *catastrophic*) failures. The device might quit working entirely if a parameter drifts too far out of specification. This category of fault is usually a design fault and should be caught in development activities, not in the field. For instance, one electric generator on the market (of which I happen to be an unwitting buyer) is powered by a one-cylinder diesel engine that must run at 3600 rpm for the generator to produce 120 V ac at 60 Hz. However, this is too fast for this particular engine design, and whenever the casting is slightly asymmetric from production, the engine in such a unit will (in the field) break crankshafts. The only recourse for repair is to change the design. Built-in diagnosis should not have to include this category of faults, and it can be assumed that the design of the object under diagnosis is "known good" in its design.

Structure, Behavior And Function

To further describe diagnosis, we need some concepts, labeled by the following words:

- *structure*: what a thing *is*, in terms of elemental components and their interrelations (in structural language)
- *behavior*: what a thing *does*, in terms of structure and behavior (in behavioral language)
- *function*: what a thing *is for*, in terms of behavior and function (in functional language, as specifications)

Each of these levels of description can be sufficiently complicated to require their own hierarchies to manage them conceptually. For instance, circuits are decomposed into interconnected subsystems, which allows us to abstract our thinking about system behavior to the subsystem level. At the behavioral level a video signal in a larger system can be treated as such without recourse to the various parts of its waveforms. Hierarchical decomposition is used to manage complexity.

The *structure* of an electronic object to be diagnosed is captured in the equivalent forms of a schematic diagram or netlist. In diagnosis the goal is to reason from failed function to (faulty) behavior at the electrical level to (faulty) structure. We already have good structural models of electronics which have grown out of computational circuit analysis and schematics editors. Those can be used extensively for deriving behavior from circuit structure using familiar causal circuit theory, expressed in waveform language, where a *waveform* is defined as an electrical function of time. However, what is missing from SPICE modeling is *functional* knowledge of the circuit. SPICE computes behavior from structure and is oblivious to function.

Computer circuit analysis has a place in the larger scheme of a KBD, though we might want to instead analyze circuits using a structural language that better accommodates function. Gerald J Sussman wrote such a symbolic circuit reasoner, called EL, back in the 1970s at the MIT AI Lab. It analyzed circuits by decomposing them functionally, identifying parts of circuits by their function, such as voltage dividers. Then it pieced together a causal explanation of how the circuit worked, like the kind of explanation an engineer would offer. His effort was part of a larger research direction into computationally-based reasoning about physical systems. (See the MIT Press book, *Qualitative Reasoning About Physical Systems*, edited by Daniel G Bobrow, 1985, noting in particular the introductory chapter and also the chapters: "How Circuits Work" by Johan deKleer, "Diagnostic Reasoning Based on Structure and Behavior" by Randall Davis, and "The Use of Design Descriptions in Automated Diagnosis" by Michael Genesereth.) These efforts focused on deriving the function of circuits from their structure, a large and uncompleted task in itself.

Functional knowledge is goal-oriented and expressed in the language of behavior. It is not the organization of behavioral knowledge, though functional theories are expressed in the language of behavior. The purpose for objects is to perform according to specification, which is a functional description of them. The function of an object constrains its possible behaviors by means of a given structure. Which of the many possible behaviors an object can exhibit is in itself an abstraction from behavior that has its own patterns and principles, those of function. Computers and televisions both exhibit electronic behavior but each is constrained to those behaviors that accomplish their different functions. The constraints on behavior involve a different kind of knowledge, that of function. Design is essentially the task of deriving structure from function. Diagnosis similarly derives what is faulty in the structure from the given description of function.

These three levels of abstraction in electronics each require a computational theory expressed in the language of the lower level. A theory of function is expressed in the language of circuit behavior. Circuit behavior is expressed in the language of structure and the behavioral laws which are circuits laws. The identification of waveforms and extraction or determination of their properties is behavioral analysis. Circuit simulators reduce behavioral laws to a fixed form that can be implemented numerically using matrix math acting upon netlists. They then output waveforms (or properties of them) that best reveal the behavior of the circuit structure analyzed.

Knowledge-Based Diagnosis

To build a knowledge-based diagnostician (KBD) requires that we design a computationally-compatible hierarchical language, or set of languages, in which to adequately describe a model of the object to be diagnosed and whatever else is in its conceptual domain. The domain of diagnostic knowledge that we want to capture can be envisioned as a loose hierarchy of four levels of conceptual abstraction, as shown below, each instantiated as a knowledge-based analyzer for its domain of knowledge.

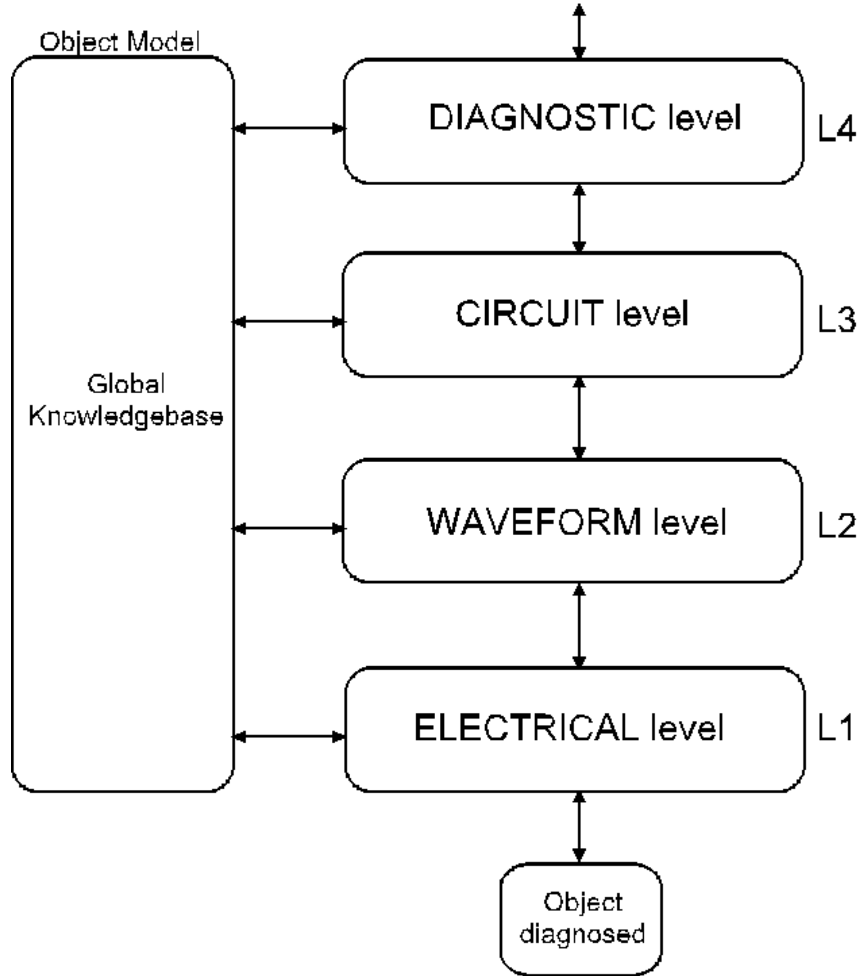
For a general KBD, the top level of abstraction involves reasoning about diagnosis itself. With a KBD, self-test might proceed roughly as follows: the diagnostic analyzer verifies the object functional model consisting of a list of specifications; each specification becomes the argument of a verify command, as suitably decomposed by L4; this circuit analyzer is more than SPICE because it also must command data acquisition, including object settings or configurations to the test level; returned to it is the result of tests.

The electrical analyzer, L1, executes tests and reasons about hardware configurations, scaling, and other test-oriented knowledge that a user of test equipment would apply, or as found in ATE programming. A *test* consists of two steps:

1. set-up (configuration of settings)
2. measurements

The second step passes the acquired waveforms to the waveform processor, L2, which extracts requested properties from the waveform, including relationships between waveforms. The set-up is derived from reasoning about instrument settings needed to acquire the waveforms. If the configuration is incorrect -- if, for example, over-range occurs on an acquisition channel -- the electrical analyzer will use this feedback to correct the scaling and try again.

The electrical analyzer is hardware-dependent and carries out commands from the waveform analyzer, L2, to acquire and generate waveforms. It examines the property-lists of waveforms in the waveform knowledgebase to determine how to acquire or generate them and then matches these requirements to the hardware instrument resources available, configures them, and proceeds to generate and acquire waveforms. The T&M (instrument) resources are described by a set of rules that correspond to the actual built-in test hardware.



The waveform analyzer responds to requests for waveform information by the circuit analyzer. For example:

IS THE BANDWIDTH OF THE OUTPUT > 5 kHz ?

or more simply,

WHAT IS THE BANDWIDTH OF THE OUTPUT ?

A waveform language might express these questions in the syntax:

Bandwidth(Output) > 5 kHz

which returns a boolean (true-false) value. The second request,

Bandwidth(Output)

returns a scalar value with units.

Often, time relationships between features of two waveforms are of interest. For instance, the voltage of a ramp might be requested at the rising edge of a gate waveform. This might be expressed as:

Voltage(ramp, when(gate(rising edge)))

The waveform analyzer needs a temporal language and rules for reasoning about waveforms in time. Relational words like *when*, *during*, *before*, and *until*, and objects such as time instants (points) and intervals are expected in such a language. L2 must also be able to extract a waveform feature such as a rising edge of a pulse, then determine in some coordinated time scale among acquired waveforms when it occurred, find that instant in the ramp waveform and extract its voltage. The waveform analyzer with L1 would be quite useful as a programming-oriented extension of the waveform extraction features found on many DSOs.

The circuit analyzer makes requests for waveform features or operations. It can pass its own simulated results based on the object model in its knowledgebase to the waveform analyzer, then compare them to measured results and pass relevant properties of the comparison to the diagnostic analyzer.

The diagnostic analyzer drives the lower levels by taking a specification of object behavior and reasoning about how to verify that behavior in the context of circuitry. It can call upon the circuit analyzer to model requested circuit behaviors and also compare them with measurements, return relevant results, and then reason about the cause of discrepancies. The diagnostic analyzer can also be given a database of statistical information about failures acquired over the history of the product to use in resolving ambiguous diagnostic decisions.

Conclusion

The self-diagnostic scenario painted here is rather grand but more feasible to implement now than it was 25 years ago when I was first envisioning it. One development path to it is for a company or business unit to work out an overall scheme with first iterations of the four languages. This sets out a framework for the grand goal. Then start at the lowest level and work upward, implementing each level as a product for self-test. The electrical analyzer reasons about testing and is hardware oriented. This is a modest but significant step toward a KBD that can find use in products today. It is essentially a reason-driven driver for embedded test hardware.

The waveform analyzer is then added. This is also an attractive product for embedded testing because it goes well beyond the limited capabilities of DSO-style waveform feature extraction, such as finding peak or rms voltages, or pulse widths. It also performs time-related functions on multiple waveforms. This extended waveform processing power brings to the user a higher level of abstraction in which to handle waveforms, allowing greater concentration upon how the waveforms relate to the corresponding circuits.

Next, the circuit analyzer is added, drawing upon SPICE for model-driven waveform prediction. The model of the circuitry in the object must be expressed in a suitable form of netlist, which can be derived directly from the design schematics. By adding the L3 level, a truly powerful KBD begins to emerge, even without L4. Diagnostic trees as they currently exist can be automated by having them command the circuit analyzer to return Boolean comparisons of measurements with those derived from the model-based SPICE analysis. These decisions then drive tree traversal.

Finally, to dispense with the manually-created trees, the diagnostic analyzer is presented as the fourth and crowning product. It uses general circuit diagnostic knowledge in the form of a rule-base and applies it to the given circuit diagram (probably in netlist form) to develop a fault search strategy. It then uses the circuit analyzer to carry out diagnosis. Its diagnostic capability would probably include AI algorithms similar to those used for game-playing such as chess. The analyzer would reason from SPICE simulations several steps ahead of the current measurement-set about which measurement would be best to do next, then execute it through the circuit analyzer. The goal of the game would be to minimize measurement costs and time for diagnosis. Each measurement is like a move on a chessboard, which reveals more about the unknown object under test while also risking additional cost and time. AI goal-search algorithms, such as an optimizing one called A*, might also appear in the diagnostic analyzer.

Why has all this not been done yet? How many electronics test engineers know much about the relevant areas of AI? How many T&M instrument design managers think much about any of this

-- or vice-presidents of engineering at major instrument or ATE companies? How many IC designers have thought this far about extending JTAG to analog test? The answer to all of these questions is: not many. Still, I would like to do this. It would be a great, and eminently possible, accomplishment heralding a new phase in electronics test and in how electronic products are designed and maintained in use.

Epistemological Postscript

Some astute electro-philosophers will have noted my use of *knowledge* where *information* might have been more accurate. I am aware of the important distinction, as most eloquently expressed by scientist-philosopher Michael Polanyi, in his classic work on epistemology (that is, on how we know) from the viewpoint of a scientist, titled *Personal Knowledge: Towards a Post-Critical Philosophy*, published by both the University of Chicago Press and Harper Torchbooks. Contrary to the misreadings of some, Polanyi, who is often cited in physical chemistry textbooks and whose son, John, won the Nobel Prize in chemistry, was hardly new-age or post-modern, and was primarily concerned with how finite and fallible human minds can transcend subjectivity to know anything as objectively true, as we believe we do in science. He emphasized that it is *persons* who know. If a machine were to know, then it would be an instantiation of a person. Information becomes knowledge when we assimilate it into the conceptual structure of our minds.

To talk about *knowledge-based systems* consequently seems to grant too much personage to what are unarguably impersonal systems. I accept that fact that the grand claims of AI in the 1960s never materialized. There are no intelligent machines. I retain use of the word *knowledge* in the same sense that we call a test instrument a logic *analyzer*. This kind of instrument does not analyze at all: its user does. Similarly, the knowledgebase of a KBD is the designer's knowledge. Nevertheless, it is used by the KBD in a very primitive way like our minds use knowledge. There is perception through processing of acquired raw data (waveforms) at various levels of abstraction, and primitive attempts to *see* the parts in the whole, through reasoning about the parts in a larger context. This ability is very limited, and in actuality only exists, as described here, as an unexploited possibility for self-diagnosis. Therefore, my use of *knowledge* to apply to KBDs is guarded, qualified, and prolusory, and should be understood in that spirit.