

## Self-Diagnosis of Electronics with Analog Circuits

*Developmental Concepts*

*by Dennis L Feucht*

Having set forth a strategy for electronic product self-test, including built-in diagnostic capabilities, in the *Introductory Concepts* [http://www.analogZONE.com/col\\_0101.htm](http://www.analogZONE.com/col_0101.htm) TechNote, this subsequent TechNote continues by formulating the next level of development detail.

The previous knowledge-based diagnostician (KBD) architecture showed four conceptual levels of abstraction, corresponding to four progressive self-test products. Each hierarchically-higher level is a superset of the lower ones, and builds upon them. Consequently, a product development path for embedded self-diagnostics begins at the lowest and conceptually most concrete level and expands upwards. These levels are necessary to manage conceptual complexity through modular construction of successively greater functional levels.

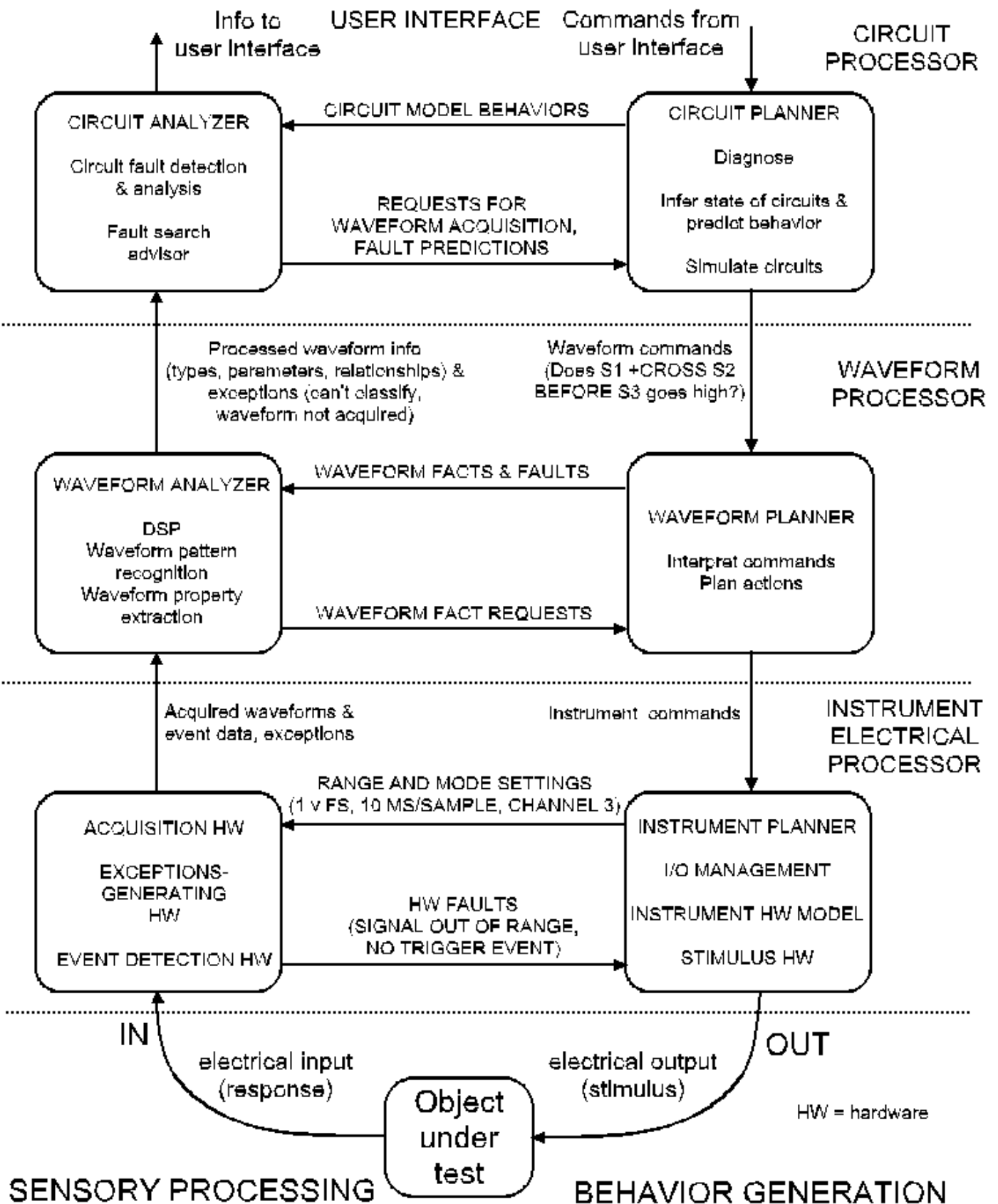
A more detailed architecture for the embedding instrumentation of self-test is shown below. The fourth level given before, that of diagnostics, is merged into the circuit level. The architecture consists of two hierarchies, one which processes electrical inputs from the object under test, and the other which commands planned electrical outputs to this object. Each hierarchy is decomposed into functional modules at the three conceptual levels, as shown. Each level is also modularized by constraining interactions between the two hierarchies to the same levels. Each of the modules in the architecture can internally be decomposed into its own subsystem hierarchy. For example, the waveform analyzer top-level functions each decompose into lower-level modules, such as the various DSP functions or pattern-recognition algorithms.

The diagram shows the major directions of information flow. The behavior-generating hierarchy is model-driven or top-down and the sensory-processing hierarchy is data-driven or bottom-up. Interaction between planner and analyzer at each level aids plan generation and execution. The planner guides the analyzer in what kind of analysis to perform and uses analyzer results to guide its planning. Feedback from analyzer to planner at each level distributes control. The model-based information of the planners gives a context for the analyzers. The analyzers report discrepancies between model and measurement. This scheme was originally conceived by James Albus of NBS (now NIST) for use in robotics.

One task is to design the languages used at the interfaces between levels. We need an instrument electrical language for generating and acquiring waveforms at the lowest level. These already exist as ATE test languages, and also as programmable instrument command-sets such as are found among GPIB equipment. What pushes the current art is that the instrument controller is given knowledge-based reasoning capabilities for solving the problems posed to it in the form of electrical-level commands. Its knowledgebase consists of a set of test rules, rules that define the instrument acquisition and generation resources available to it, and rules of configuration and interaction of these resources. With programmable hardware, the instrument controller then configures the instrumentation according to the plan it has devised by reasoning from its electrical rule-base.

We also need a waveform language. Some have been developed in AI research, which is a place to start. As electrical functions of time, waveforms involve temporal reasoning. The waveform language must include semantics complete enough to capture all essential temporal relationships within and between waveforms. The waveform analyzer must not only perform existing property extractions, such as finding peak, average, or rms values, pulse widths, or overshoot, but also new ones needed in a circuits context, such as waveshape identification (which is also needed to determine the validity of deriving pulse parameters such as pulse width and risetime), ramp slopes, sine distortion, envelope decay, event times such as zero-crossings, peaks, burst gatings, and video, network, or multiplexed waveform decompositions.

# Hierarchical Instrument Architecture, Computational Model



Some emergence of waveform language has necessarily occurred in SPICE and other circuit and system simulators. However, to support a KBD, what has been done by humans will need to be done by the KBD. This will require a richer repertoire of waveform processing than is found in simulators. AI work on qualitative reasoning about circuits also lays the groundwork for symbolic (versus numerical) computation of circuit behavior from function, and the identification of function from structure. Ideally, all the KBD needs is a schematic diagram from which it computes a diagnostic plan. That step is probably too large for the first generation of KBDs. Instead, the circuit-level processor is given a set of general diagnostic rules and rules about function worked out manually by a knowledge engineer, an *expert systems* expert. An embedded KBD need only compile the rules and routines needed for the particular object of diagnosis, to minimize on-board KBD memory requirements. The circuit planner has SPICE within it for computing expected waveforms. The analyzer does fault search and detection for identifying relevant waveform discrepancies between model-calculated and object-measured waveforms. Its fault advice guides the planner in what to do next.

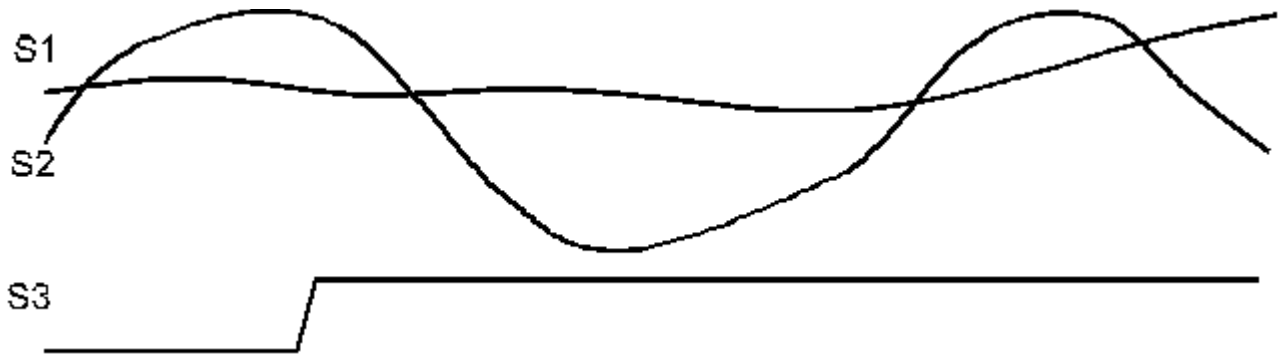
### **Electrical (Instrumentation) Processor**

The lowest conceptual level is nearest the object of test and interfaces with it. The instrumentation used to stimulate and acquire data from it is optimized for the object, because the object in self-test is the rest of the product, which is known and fixed in what it is. In many cases, mixed-signal waveform generation and acquisition constitutes a mini-ATE using some of the non-test systems already in the object of test. On the sensory processing side, a data acquisition subsystem needs not only to acquire sampled values of voltage or current, but also detect exceptions, such as out-of-range values or malfunctioning ADCs. These are reported to the behavior generating electrical module, which uses a rule-base to plan what to do next. For example, an over-ranged waveform fires a rule with the action of changing the PGA at the ADC input to a less sensitive scale for that channel.

The electrical processor must do more than this simple task, however. It also must plan instrumentation configurations for achieving a commanded measurement, and then set the instrument parameters accordingly. If the configuration is faulty, exceptions processing hopefully will correct it. A well-designed rule-base minimizes such occasions. Part of what helps the electrical processor correctly configure a measurement is model-driven knowledge of what to expect in acquired waveforms. This aids, for instance, in setting the sample rate to avoid aliasing and maximizing waveform time resolution. Model-based information passes through from the circuit processor (which has acquired it from simulation by the circuit analyzer) as part of the information about the waveform to be acquired. The waveform analyzer uses it to set the correct sample-rate parameter in the ACQUIRE command sent to the electrical processor. Instrument planning is thereby given an implicitly larger contextual framework in which to carry out commands, though the electrical level knows of nothing beyond the electrical: voltages and current, sample rates, instrument settings and characteristics. Yet it is guided in its activity by a wider scope of understanding at the higher levels through the specific requests made of it.

## Waveform Processor

Simulators have some waveform analysis capability, though it is not sufficient for KBD. The waveform analyzer must not only extract properties from waveforms, but also identify which kind of waveforms they are, align them in time and extract properties of them based on other waveforms. For instance, suppose two signal waveforms, S1 and S2, are compared in value in an interval of time after the leading edge of digital signal S3, as shown below.



The language of the waveform processor must be able to support questions such as: Does S2 cross S1 with a positive slope after S3 goes high? The waveform planner parses them and plans how to command the instrument processor to answer the question. The waveform analyzer is informed about what is expected (for instance, that S3 is digital) and is given the goal of extracting the answer and passing it to the circuit analyzer. If waveform analysis finds that S3 is not digital, the waveform planner probably cannot do anything about it. The circuit planner has probably misidentified S3 and will be informed by the circuit analyzer.

Design of the waveform processor is a large task beyond conventional ATE, no less self-test. The task begins with development of a waveform language. The first subtask is to identify and represent what the domain of the language - waveform and electrical - has to express. One hack at such a list is given below. It is certainly not refined but is intended to give the flavor of what might be in the language of waveforms.

### Waveform and Electrical Object Representation

Lists are denoted by parentheses: ( ... )

Object representations are of the form: object-name list-pointer-symbol list

Symbol Key:

| disjunction (logic *or*)

n number

i interval pointer

w waveform pointer

t skew pointer

e event pointer

f boolean flag

r rule

waveform  $w \rightarrow$  (WAVEFORM waveform-name property-list)

*properties*

sample-array: waveform unit and sample-value array pointer

sample-rate:  $1/\Delta t$  between samples

amplitude-range: full-scale value of waveform array

trigger-event: trigger event list pointer

composition: composition of waveform from other waveforms

*composition operators*

+: add two waveforms

-: subtract two waveforms

seq: sequence piecewise in time

fm: frequency-modulate one waveform with other

am: amplitude-modulate one waveform with other

pm: phase-modulate one waveform with other

interval  $i \rightarrow$  (INTERVAL beginning-index ending-index waveform pointer)

time-skew  $t \rightarrow$  (SKEW ref-waveform-pntr waveform-pntr time-skew)

event  $e \rightarrow$  (EVENT event-name event-sequence)

*event operators*

OR: disjunction of events - OR( $e_1, e_2$ )

AND: conjunction of events - AND( $e_1, e_2$ )

SEQ: sequence of events - SEQ( $e_1, e_2$ )

Operators can be combined in trees; for example: (SEQ( $w, (OR (AND b, c), AND(y, z)))$ )

Primitive events are:

+CROSSES( $w_1, w_2 | n$ ) -  $w_1$  crosses  $w_2$  or  $n$  with slope  $\geq 0$

-CROSSES( $w_1, w_2 | n$ ) -  $w_1$  crosses  $w_2$  or  $n$  with slope  $\leq 0$

DELAY( $e, n$ ) - delay by  $n$  from  $e$

COUNT( $e, n$ ) - count  $n$  occurrences of  $e$

FREQ( $e, f_1, f_2$ ) - frequency of  $e$  is between  $f_1$  and  $f_2$

PERIOD( $e, f_1, f_2$ ) - period of  $e$  is between  $f_1$  and  $f_2$

+WIDTH( $e, t_1, t_2$ ) -  $e$  is high for a time between  $t_1$  and  $t_2$

-WIDTH( $e, t_1, t_2$ ) -  $e$  is low for a time between  $t_1$  and  $t_2$

$>(w, n)$

$\geq(w, n)$

$<(w, n)$

$\leq(w, n)$

$=(w, n)$

MAX( $w$ ) - maximum of  $w$

MIN( $w$ ) - minimum of  $w$

EXTR( $w$ ) - extremum of  $w$

module m → (MODULE module-name property-list)

rule r → (RULE (IF (conditions) THEN (actions)))

Then with these semantics a waveform command set follows.

(arguments of commanded function → result)

Commands:

(n|i1|w1 i2|w2 → i3)

+CROSSES returns interval during which i2|w2 crosses n|i1|w1 with a positive slope

-CROSSES returns interval during which i2|w2 crosses n|i1|w1 with a negative slope

> returns interval where n|i1|w1 > i2|w2

< returns interval where n|i1|w1 < i2|w2

≥ returns interval where n|i1|w1 ≥ i2|w2

≤ returns interval where n|i1|w1 ≤ i2|w2

= returns interval where n|i1|w1 = i2|w2

(i1|w1 i2|w2 → i3)

BEFORE returns interval of i1|w1 that occurs before i2|w2

UNTIL

AFTER returns interval of i1|w1 that occurs after i2|w2

SINCE

WHEN returns interval of i1|w1 that overlaps in time with i2|w2

DURING

CORR returns correlation of i1|w1 and i2|w2

CONV returns convolution of i1|w1 and i2|w2

+ returns sum of i1|w1 and i2|w2

- returns difference of i1|w1 and i2|w2

× returns product of i1|w1 and i2|w2

/ returns quotient of i1|w1 and i2|w2

(i1|w1 i2|w2 → t)

SYNC returns time-skew between i1|w1 and i2|w2

(i1|w1 → i2)

BECOMES returns time interval which contains as beginning and ending values the beginning value of i1|w1

GOES

HIGH returns interval of i1|w1 during which it satisfied the conditions for being high, found on the property-list of w1 or the waveform pointed to by i1

LOW - same as HIGH except for low property

+SLOPE returns interval during which the slope of i1|w1 ≥ 0

-SLOPE returns interval during which the slope of i1|w1 ≤ 0

MAX returns interval during which i1|w1 is at maximum value in i1|w1

MIN returns interval during which  $i1|w1$  is at minimum value in  $i1|w1$

EXTREMA returns MAX | MIN

INFPT returns interval where inflection points of  $i1|w1$  occur

SINE - returns interval in which waveform recognizer classifies  $i1|w1$  as having this waveshape

SQUARE

TRIANGLE

VIDEO

SEGMENT returns interval of endpoints between waveshapes

( $n1\ i1|w1 \rightarrow n2$ )

VALUE returns value of  $n1$ th sample of  $i1|w1$

( $i1|w1 \rightarrow n\ f$ ) false flag (0) indicates failure to produce  $n$

DURATION returns time duration of  $i1|w1$

FREQ returns frequency of  $i1|w1$

PERIOD returns period of  $i1|w1$

SLOPE returns slope of  $i1|w1$

AVG returns average value of  $i1|w1$

RMS returns rms value of  $i1|w1$

SPAN returns peak-to-peak value of  $i1|w1$

NLIN returns nonlinearity of a monotonic  $i1|w1$

THD returns total harmonic distortion of a sinusoid  $i1|w1$

CROSSINGS returns number of +CROSSES and -CROSSES intervals in  $i1|w1$

( $i1|w1 \rightarrow f$ )

AC returns flag indicating whether  $i1|w1$  is varying (dynamic)

DC returns flag indicating whether  $i1|w1$  is constant (static)

( $i1|w1 \rightarrow w2$ )

FFT returns FFT of  $i1|w1$

PSD returns power spectral density function of  $i1|w1$

PDF returns the probability density function of  $i1|w1$

AM extracts modulating waveform from  $i1|w1$

FM

PM

SQUARE returns  $i1|w1$  squared

SQRT returns square root of  $i1|w1$

ABS returns absolute value of  $i1|w1$

( $n\ i1|w1 \rightarrow w2$ )

DER returns derivative of  $i1|w1$

INT returns integral of  $i1|w1$

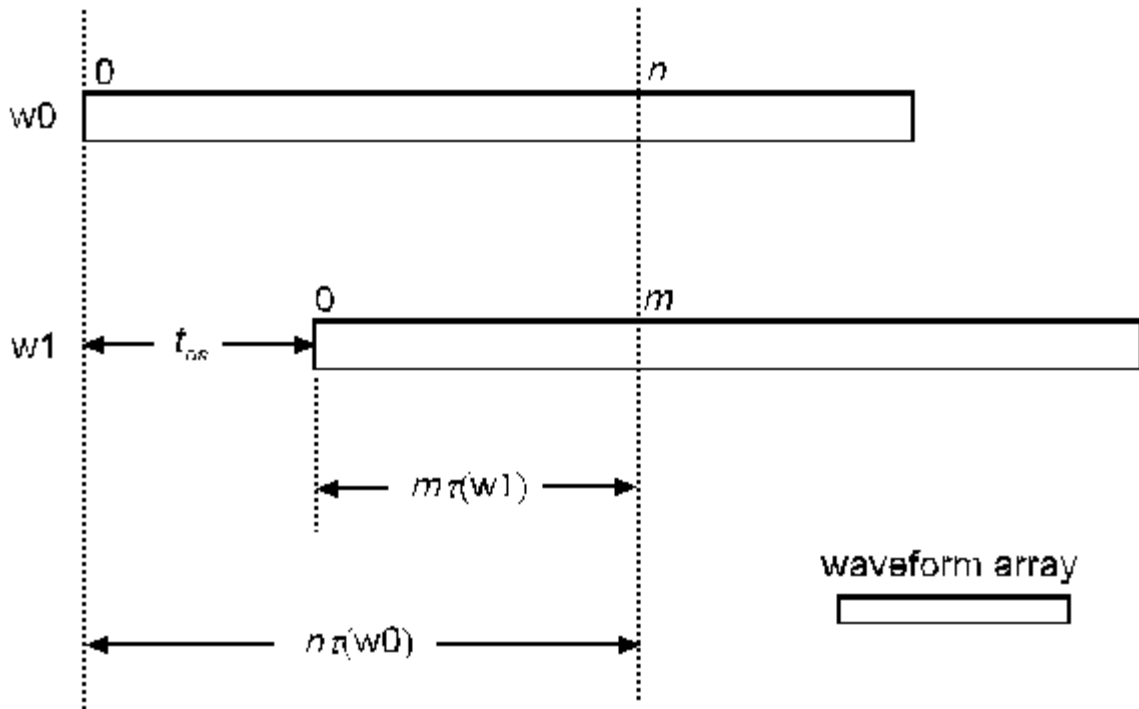
LPF returns low-pass-filtered  $i1|w1$  at break frequency  $n$

HPF returns high-pass-filtered  $i1|w1$  at break frequency  $n$

( $n1\ n2\ i1|w1 \rightarrow w2$ )

BPF returns bandpass-filtered  $i1|w1$  between break frequencies  $n1$  and  $n2$   
 BRF returns band-reject-filtered  $i1|w1$  between break frequencies  $n1$  and  $n2$

One question that arises is why time intervals have been defined in the waveform language but not points in time. We certainly are interested in exactly at what times some electronic events occur. The reason has to do with the discrete sample values in the array of waveforms. Consider two waveforms  $w0$  and  $w1$ .  $w1$  is offset in time from  $w0$ , both represented below by an array of sample points with the index of each starting at the left ends.



$\tau(w)$  is the sampling period of  $w$ , and  $m$  and  $n$  are waveform array indices. From the graph,

$$n \cdot \tau(w0) = t_{os} + m \cdot \tau(w1)$$

Also, let:

$$m' = \frac{n \cdot \tau(w0) - t_{os}}{\tau(w1)}$$

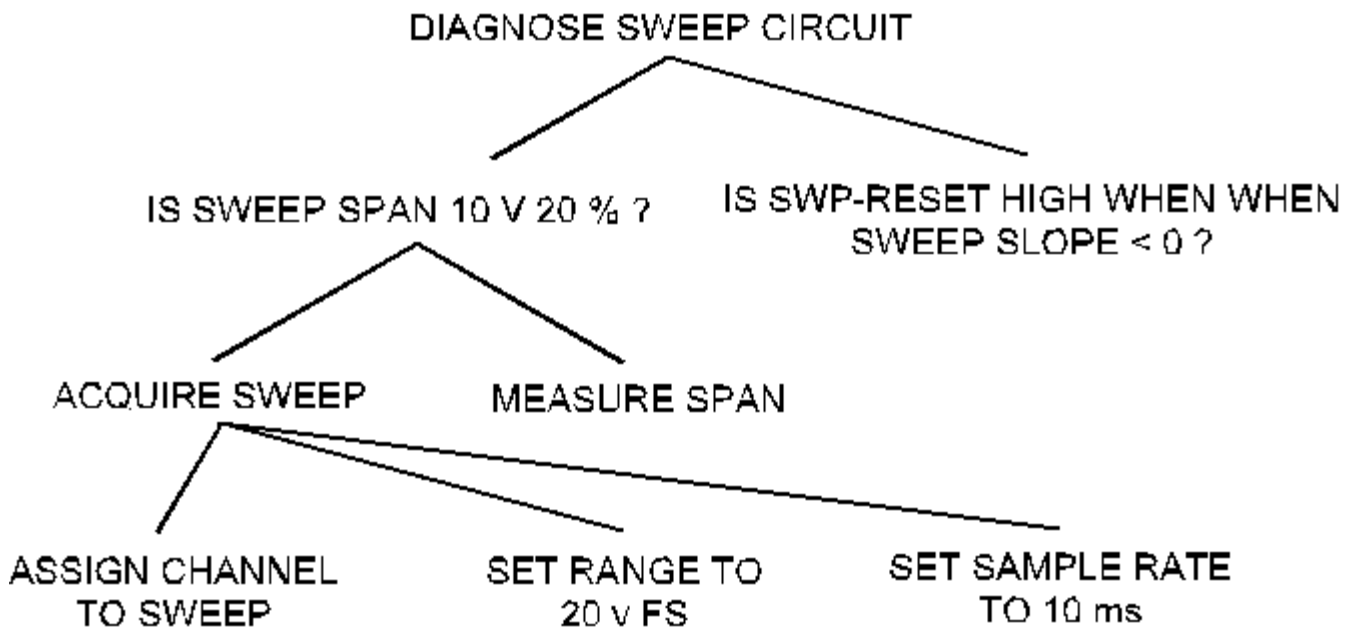
Then  $m'$  is a continuous function because  $t_{os}$  is continuous. However,  $m$  is discrete. Consequently,  $w0(n)$  is *not* necessarily simultaneous with  $w1(m)$  but corresponds in time with the closed interval  $[nb, ne]$ , where  $nb$  = least integer of  $m'$  and  $ne$  = greatest integer of  $m'$ .

A mixture of continuous and discrete time quantities results in somewhat more complicated math for time calculations, resulting in the use of time intervals as a basic time quantity. A point in time is simply an interval with equal endpoint values.

## Task Decomposition

The command hierarchy is characterized by multilevel task decomposition, an example fragment of which is shown below for diagnosing an analog oscilloscope sweep subsystem. Perhaps the user-level fault symptom is no trace on the screen, only a bright dot at the left side. The user provides the fault symptom, as observed, to the circuit processor. The planner invokes its inference engine to conclude that it should determine whether there is a sweep waveform at the output of the sweep generator. It runs a simulation on the sweep subsystem, and sends the sweep output waveform to the waveform processor with the request to return the span (pk-pk value) of it. The waveform planner commands the electrical planner to acquire the waveform. It configures the sweep circuit and also the self-test ADC multiplexer to input the sweep waveform on a particular channel, sets the scaling and sample rate, and signals the acquisition hardware to arm its triggering. Once the waveform is acquired and passed to the waveform analyzer, the span is extracted and sent to the circuit analyzer. If it is within the acceptable bounds, then the cause of the problem is not a lack of sweep waveform from the generator. The circuit planner deduces the next move; perhaps it is in the sweep reset circuit...

### Hierarchical Task Decomposition



## Closure

Much is left to be thought out in the design of this diagnostic architecture. Additional challenges lie in reducing it to embedded status for self-test. Yet by the measure of required computing throughput per unit cost, it appears feasible to start (actually, restart) a project to develop a product line of the three processors, beginning at the bottom level and working up. It could be the basis for a company or business-unit start-up plan. Anybody want to do it?

The most likely companies to come to mind are the T&M instrument companies such as Tek and Agilent, and leading ATE companies. However, vision into the possibilities of AI-style computing (no less robotics) for self-test is limited in the T&M world because it is such a different world. This is an interdisciplinary quest, requiring both good analog and digital circuits engineers and also good robotics-AI software engineers. It takes a management team with wide, multidisciplinary vision.

More likely than not, a software- or computer-oriented company will eventually do this. The electronics know-how is essential to project success, but most of the work is in the software. Once a company has a growing knowledgebase from which to customize reduced KBDs for products, nearly any kind of electronic product costing over \$2000 US with an embedded processor in it is a likely candidate for KBD self-test, especially those easily repaired once the fault is located. The market size is large enough for KBD.

