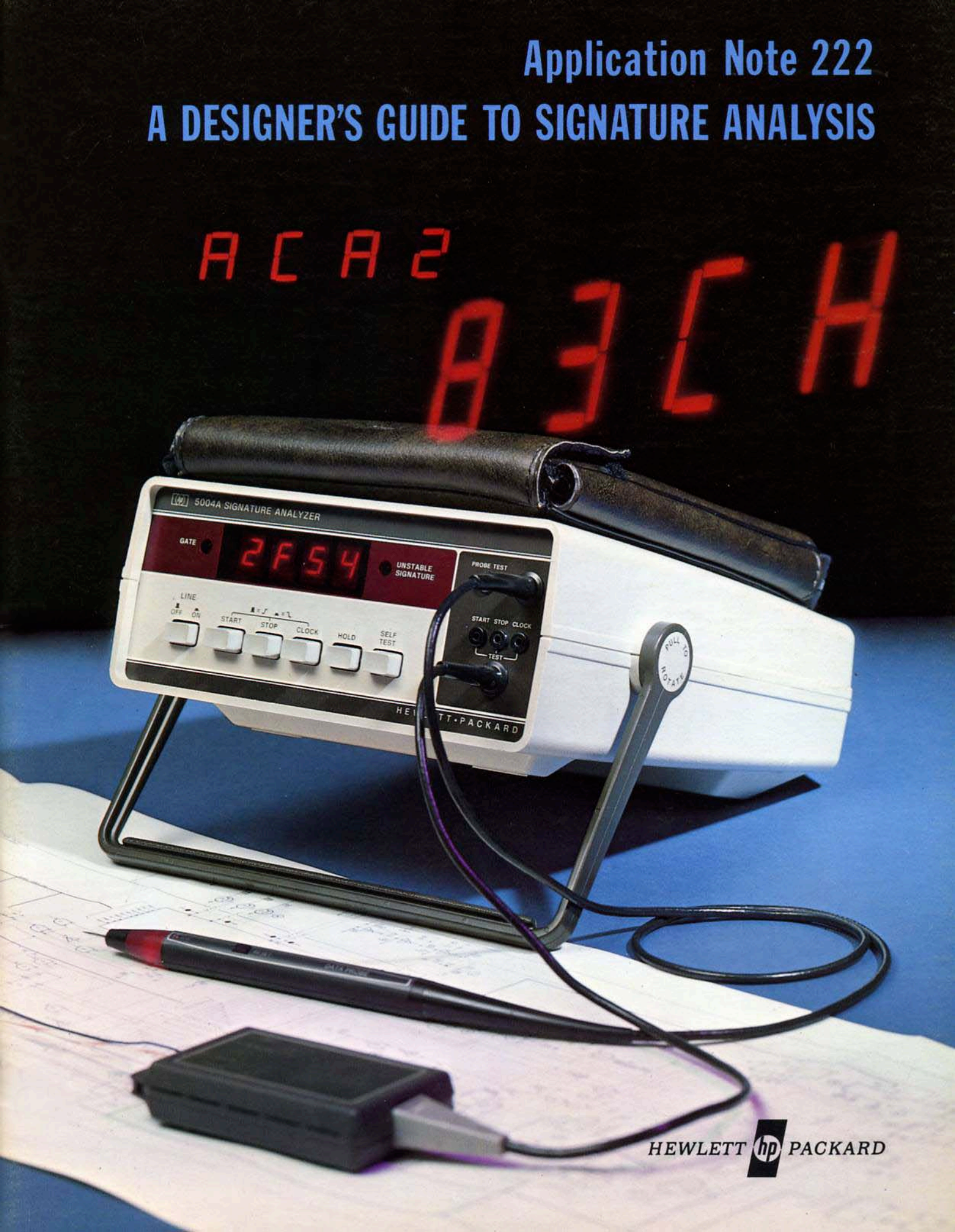
Application Note 222

A DESIGNER'S GUIDE TO SIGNATURE ANALYSIS



HEWLETT hp PACKARD

Application Note 222

A DESIGNER'S GUIDE TO SIGNATURE ANALYSIS

TABLE OF CONTENTS

Section I
Introduction to Signature Analysis

SCOPE

The intent of *A Designer's Guide to Signature Analysis* is to provide product designers and service engineers with the information needed to understand and then apply the principles of Signature Analysis (SA) to digitally based products. This is accomplished by describing: the fundamental concepts of SA, the pertinent features and use of the Signature Analyzer, how to design products for testability, specific SA design techniques for various classes of circuits, and documentation strategies. Examples are provided to help illustrate the topics.

Major emphasis is placed on applying SA to microprocessor based systems because this is where the greatest benefit is most often realized. ROM based and other logic structures are also discussed since SA can often be a cost-effective vehicle for enhancing their serviceability.

Other available literature relating to, or not covered in, this application note are: the 5004A Signature Analyzer Data Sheet and its Operating Manual, *"Electronics Magazine,"* March 3, 1977, Vol. 50, No. 5, p. 89, and *"Hewlett-Packard Journal,"* May, 1977.

THE NEED FOR SIGNATURE ANALYSIS

The current trend in digital system design is toward bus-structured machines that rely heavily on Large Scale Integration (LSI) components, (e.g., microprocessors, ROMs, RAMs, complex interface devices). By controlling the communication and algorithmic interaction between bus devices, much of the dedicated hardware logic previously associated with performing complex signal and data processing tasks yields to software data munipulation. However, when logic signals are replaced by data bit streams in, say, a microprocessor system, the functional operating characteristics of the circuit are not necessarily associated with specific hardware components and are much more difficult to characterize and define because of the long complex data streams that are present. This is further complicated by the fact that many of the devices connected to the bus are bidirectional, (i.e., can both input and output data on the same bus pins).

Unlike random logic circuits, all but the most trivial microprocessor system defy fault analysis unless heavily supported by documentation and extensive use of circuit isolation techniques. This is why most of the servicing of the microprocessor portions of products has been handled by board exchange programs; and why most service people are wary of microprocessors. Even when an experienced service technician is armed with logic analyzers and other specialized service and design instruments, finding faulty components in a microprocessor or bus structured product requires detailed knowledge of the circuit and can be very time consuming.

The main problem is that test instruments may provide either too much or too little information. What is needed to afford component level field repair, is a means of compressing the long data streams present in a system running at its normal operating speed, into a concise, easy to interpret, and meaningful readout. The readout need not tell any more than whether or not a particular circuit node is operating correctly.

Up to now, the best means for finding component level faults was by a technique known as "transition counting," where the number of changes of logic state on a node, during a given time interval, were counted and compared to the numbers recorded for the same node on a known good circuit. But "transition counting" has practical limitations. For one, an external stimulus is often required in order to provide logic activity on circuit test nodes. Another is that a large number of transitions must occur to produce statistically usable information, since counts produced are not time dependent, (i.e., the correct number of "transitions" might occur; but at the wrong time). Refer to *Electronics,* March 3, 1977, Vol. 50, No. 5, p. 89. Because of these limitations, board swapping often took precedence over component level field repair.

Clearly, better methods to meet the service challenge imposed by the proliferation of microprocessor based products were needed.

THE CONCEPT OF SIGNATURE ANALYSIS

The basic ingredients of Signature Analysis (SA) are "data compression" and "circuit generated stimulus." Both of these features exist, to some degree, in transition counting, but in Signature Analysis they are refined in a manner that affords greater overall performance, in terms of locating faults in complex digital circuitry.

"Data compression" is achieved in the Signature Analyzer by probing a logic test node from which data is input for each and every circuit clock cycle that occurs within a circuit controlled time window. Within the Signature Analyzer is a 16-bit feedback shift register into which the data is entered in either its true or complement logic state, according to previous data-dependent register feedback conditions. In all, there are 2^{16} (or 65,536) possible states to which the shift register can become set to during a measurement window. These states are then encoded and displayed on four hexadecimal indicators, ($2^{16} = (2^4)^4$), and become a "signature." This signature is then a characteristic number representing time dependent logic activity during a specified measurement interval for a particular circuit node. Any change in the behavior of this node, (e.g., even a transition that is one clock cycle late or skewed with respect to the clock) will produce a different signature, indicating a probable circuit malfunction. A single logic state change on a node is all that is necessary to produce a meaningful signature. And, because of the compression algorithm chosen, measurement intervals exceeding 2^{16} clock cycles will still produce valid, repeatable signatures.

The signal that causes the node to produce a signature is the "stimulus." In SA, the stimulus is supplied by the product itself. By doing this, a controlled environment can be created wherein selected circuit portions can be tested independently of others, while maintaining full dynamic operation. Additionally, synchronization and measurement intervals for the Signature Analyzer can be controlled by the product under test. In microprocessor systems, the stimulus is nothing more than a program, (generally in ROM) that exercises the rest of the system. Taking advantage of the data manipulative capabilities of microprocessors, generating good stimulus patterns that exercise individual devices in the product is usually not very difficult. Indeed it is often true that, the more complex the system, the greater the benefit derived from using SA. The technique can take much of the uncertainty out of servicing microprocessor and other bus structured products.

Section II
General Design Principles for Using SA

DESIGNING IN TESTABILITY

In any product, good testability doesn't happen by accident. It has to be designed in. Sometimes it can be patched on later, but with today's high density, high complexity products, this method is often very costly. Trying to retrofit the SA technique can be even more so.

Clearly, the time to start thinking about serviceability is early in the design phase, so that before the hardware design gets frozen, a repair strategy can be decided upon and provisions for its implementation made, (e.g., allocating ROM space and room for jumpers, test points, and other service hardware).

A product's overall testability can almost always be enhanced by judicious partitioning of its functional elements. For example, breaking a product's circuitry into minimally interdependent modules, each on its own plug-in P.C. board, makes the instrument easier to understand and repair than a massive jumble of interwoven functional circuits on a few large P.C. boards. Fortunately, microprocessor systems generally lend themselves well to orderly partitioning. But when SA is included in their design, large boards become much less of a servicing disadvantage, and the savings that they represent in the overall manufacturing cost of the product can be substantial.

Another useful concept is that of "self-test" or "performance verification." This usually is in the form of exercises and measurements performed by the product on itself to check out as much of its circuitry as practical. The self-test sequence can be initiated automatically whenever the power is turned on, or by pushing a test button, usually located on a front panel.

A typical sequence results in a GO/NO-GO "confidence" indication visible to the operator. Additionally, the self-test can provide some degree of failure diagnostic information if so desired. It is rarely possible or cost-effective to self-diagnose beyond the board or function level, however.

The main advantages of self-test are: 1) alerting the operator to a possible operational problem and 2) reassuring him that even though he may be having trouble getting the product to perform, it's really ok. This latter advantage can save a frustrated customer from sending a good (but probably complex) product in for unneeded service or repair.

Once a product is determined to be faulty, it must be repaired or discarded. Since most electronic products are not yet disposable, the objective is "find the fault and fix it." If the product has been designed with good serviceability in mind, most any fault can be localized in short order, and repaired. This is where the payoff can come from a good application of SA, partitioning, and documentation.

But who designs the SA capability into the product? In most cases it's a shared responsibility between the design and service engineers. The fact is there is no clear line between them when it comes to designing in good serviceability. Usually, however, the designer takes care of making sure the hardware is compatible with good SA practices, and the service engineer prepares the service documentation. Both participate in writing the test stimulus programs and overall troubleshooting strategy.

Since SA is a new concept, it follows that the learning curve principle applies to how effectively a design engineer can utilize it in a new product design. The "tricks" he picks up from his first SA application will likely make his second one easier and of even greater benefit to that product.

THE SIGNATURE ANALYZER

The Signature Analyzer employs a unique data compression technique that reduces any long, complex data stream pattern on a logic node into a four-digit signature. When placed on a logic node whose correct "signature" is known (having been determined empirically from a known good product) a comparison can be made, with the circuit running at full operating speed, to determine good or bad circuit operation. By probing various nodes, finding bad signatures and then "tracing" them back to the functional origin, the actual fault source can be found. The Signature Analyzer is more than just a transition counter, since its internal feedback shift register circuit is both data and time dependent, allowing for a signal reduction algorithm that produces highly reliable results.

In operation, signals supplied to the Signature Analyzer initiate (Start) and terminate (Stop) a measurement time period (window, gate). A Clock input synchronizes and controls the data sample rate of the probe input so that data is input to the Signal Analyzer and processed every clock cycle within a Start/Stop interval. The Start and Stop inputs are individually selectable for logic "1" or "0" levels. The Clock input is edge triggered and can be selected for either rising or falling edges.

Some General Guidelines:

1. The measurement window framed by the Start and Stop signals must be unique and synchronized to all the nodes tested so that consistent signatures are produced. The number of clock edges enclosed must be a constant for each test set-up condition.
2. Data must be synchronous with and stable during the triggering edge of the clock. Data set-up time must be included.
3. Start and Stop can be physically tied together with any of the 4 logic level select combinations permitted. This minimizes the number of connections to the test circuit, and is therefore a desirable practice.
4. Although a single Start/Stop sequence produces a valid signature, troubleshooting is faster with continuous cycling, for which guideline #1 applies.

An illustration of the timing involved in a measurement window can be seen in Figure 1. (Consult the Signature Analyzer instruction manual for speed and setup time constraints, and loading.)

The most common connection point for the Signature Analyzer Clock is the clock phase of the circuit under test where address and data lines are stable.

Start and Stop signals that define the data sampling interval can be taken from address lines, state pointers, software controlled output ports, or any other signals that identify the presence of a unique data stream for the particular set of circuit nodes of interest. Keep in mind, however, that the more circuitry involved with generating these signals the higher the probability that a fault will be present in that circuitry. Provisions should be made to allow for the verification of these individual circuit components so that any faults in them can be found.

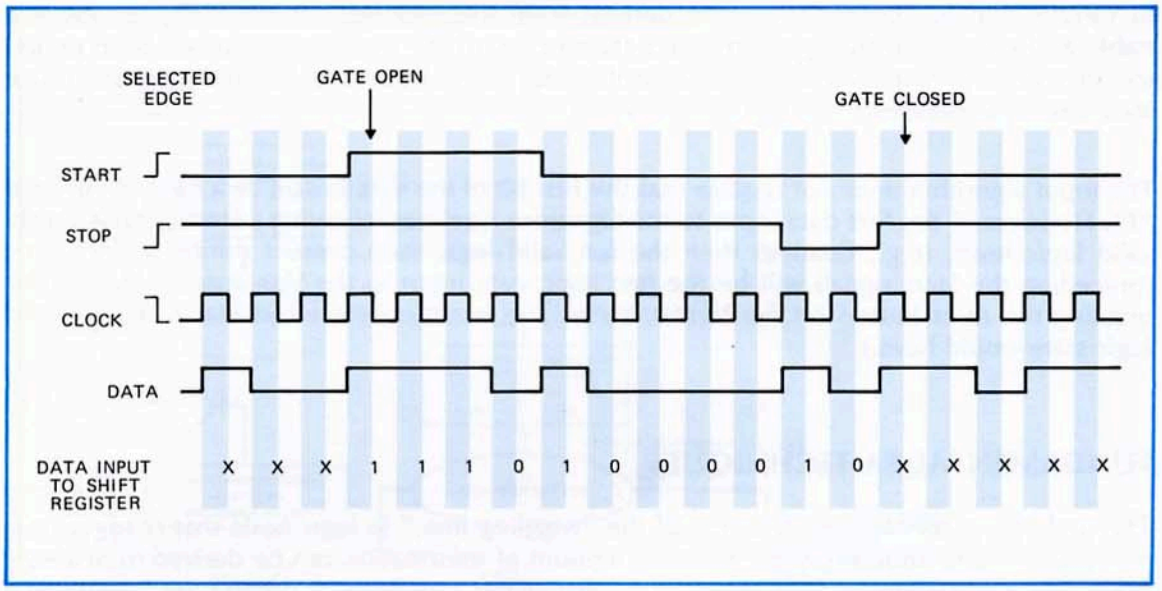


Figure 1. Typical data input to the Signature Analyzer illustrating the timing relationships between the control signals

Both the Clock and the Start and Stop signals generally remain on the same circuit points through many or even all Signature Analyzer probe readings. In this manner, much of the signature gathering process can be simply a matter of moving the Signature Analyzer input probe from node to node on the circuit under test. By minimizing the number of test setups requiring unique Start, Stop, and Clock connections and edge selections, troubleshooting effort can be reduced. The degree to which this can be done depends largely on the circuit's design structure.

The data input of the Signature Analyzer consists of a dual threshold, (logic "Ø" and logic "1") comparator circuit. In effect, these comparators sample the input at the time of the selected clock edge. To the node under test, the input looks like a 50K resistor pulled to a nominal 1.4V reference. Since 1.4V lies between the two threshold voltages, a change from an active logic level to a floating (or high impedance) condition will not cause a change in the logic state input to the Signature Analyzer. Figure 2 demonstrates how this input circuit reacts

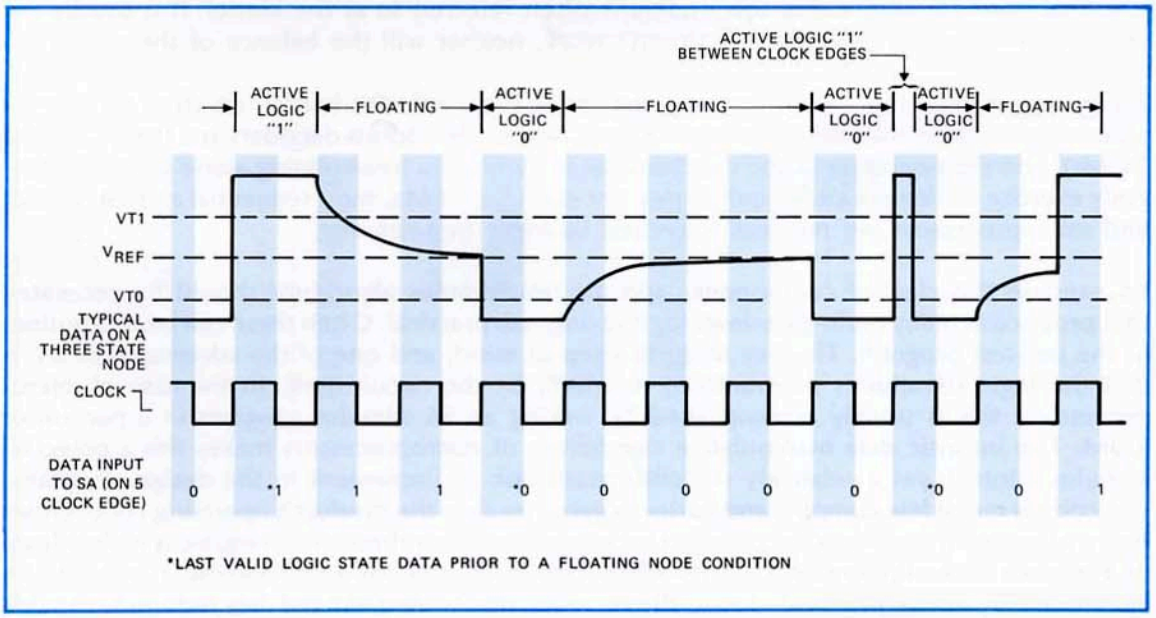


Figure 2. Floating nodes retain the logic state of the last valid data clocked in

to various signals. The major benefit derived from this technique is the ability to obtain a stable and repeatable signature from a 3-state node. (If the input had, instead, been pulled toward Vcc, stray test node capacitance could delay the transition to the high state, and cause inconsistent signatures.

The input algorithm does not require that the first bit of the data stream be a hard "Ø" or hard "1". However, if the first data input to the Signature Analyzer following a Start signal is a non-valid logic level, (e.g., floating) then the last valid logic state present during a clock time (preceding the Start signal) will be the first logic state input to the Signature Analyzer. (Depressing the reset button on the Signature Analyzer has the same effect that a "Ø" last valid logic state would have.)

FUNDAMENTAL SA TECHNIQUES

The underlying concept of SA is that of the "wiggling line," (a logic node that changes from one logic state to another). Only a limited amount of information can be derived from a logic node that never changes state during a particular test sequence. It doesn't get "exercised." This applies to nodes on P.C. boards as well as the logic nodes internal to an I.C. On the other hand, if a node actively changes state in a proper manner at the correct time for that circuit, much more valid information and confidence results. In fact, it often statistically doesn't even matter very much "what" caused it to wiggle for it to provide useful diagnostic information. However, a "truth table" exercise on the components is generally more exhaustive and will catch a greater number of failures.

This leads to the "free-running" concept. Free-running, for purposes of SA, means the following:

1. getting the circuit to run in a repetitive loop with
2. only a minimum number of the circuit's logic components required to do this, but
3. causing the maximum number of logic nodes in the circuit to wiggle.

In the case of microprocessors, controllers, sequencers, and ASM's, free-running is often accomplished by opening the data (or instruction) input bus and then forcing in an instruction (such as NOP) or control that causes a continuous cycling through the entire address or control field. The circuitry doing this cycling is often referred to as the kernel. It is usually the origin or heart of the system. If it doesn't work, neither will the balance of the circuit.

Taking signatures while in the free-running mode can verify the kernel, much of the combinatorial circuitry on the address or control lines, (especially address decoders and the data in the ROMs), and the operation of the data bus. But in most cases free-running alone cannot sufficiently exercise all devices and circuit nodes. For example, RAMs, most sequential and LSI circuits, and microprocessors are not well exercised by mere free-running.

To exercise this class of components, specific test stimulus algorithms should be generated that produce as many multi-gate-level logic changes as practical. Often these can be subroutines in the self-test program. The key thing to keep in mind, and one of the advantages of SA, is that the logic stimulus is programmed internally by the circuit itself. In the case of microprocessors, this is usually accomplished by writing an SA stimulus program in a portion of ROM. The intrinsic data manipulative capabilities of microprocessors makes this a powerful stimulus control, yet a relatively straightforward task to implement in the design. Typically, this code is much less complex and easier to generate than the product's operating code. Other types of systems, less capable of generating stimulus algorithms, may require a higher level of firmware committment. And, with many classes of random logic systems, the whole SA approach may not be practical, due to the very high hardware overhead they require to provide good stimulus and data feedback control.

Most applications of SA will likely be in bus structured systems. Bus structures can offer very definite design advantages, but can make fault analysis difficult. The tough question to answer is: which element is putting the bad data on the bus? This may not appear to be a very profound question, but it turns out that there are several major classes of faults that can make bus fault location difficult. The key things to mention at this time are the two techniques used with SA to deal with busses. These are the abilities to:

1. disable busses and bus elements, and
2. open feedback loops.

Busses will be discussed further in the section on busses.

Although the Signature Analyzer is primarily a service and production troubleshooting tool, it can prove valuable on the design bench for checking out prototype and pilot production run products, once the good signatures are known. Timing errors and excessive node settling times can sometimes be diagnosed by varying the system clock rate and observing any change or instability of the signature.

Section III

An Illustrative Example

Figure 3 is a generalized microprocessor system diagram with SA designed in. It has been partitioned into a near ideal configuration for purposes of illustrating some of the fundamentals of an SA product design. (Additional application techniques are covered in later sections.) The system contains a microprocessor with an interrupt system, 3 ROMs, 2 RAMs, a couple of I/O devices, an LSI part, and associated interfacing logic. The power supply and clock are taken for granted, since their operation is easy to verify by conventional means, (e.g., the Logic Probe feature built into the Signature Analyzer can be used as a quick check).

The procedure which follows is for illustrative purposes and does not necessarily demonstrate a preferred implementation of SA.

First of all, how does one really know whether a complex microprocessor-based system is working? Hard failures such as blank display segments and dead power supplies are easy to find. But soft failures, like data dependency errors and random glitches are more difficult. This is where a good self-test program can often provide some help. The unstable signature indicator on the SA can also be useful.

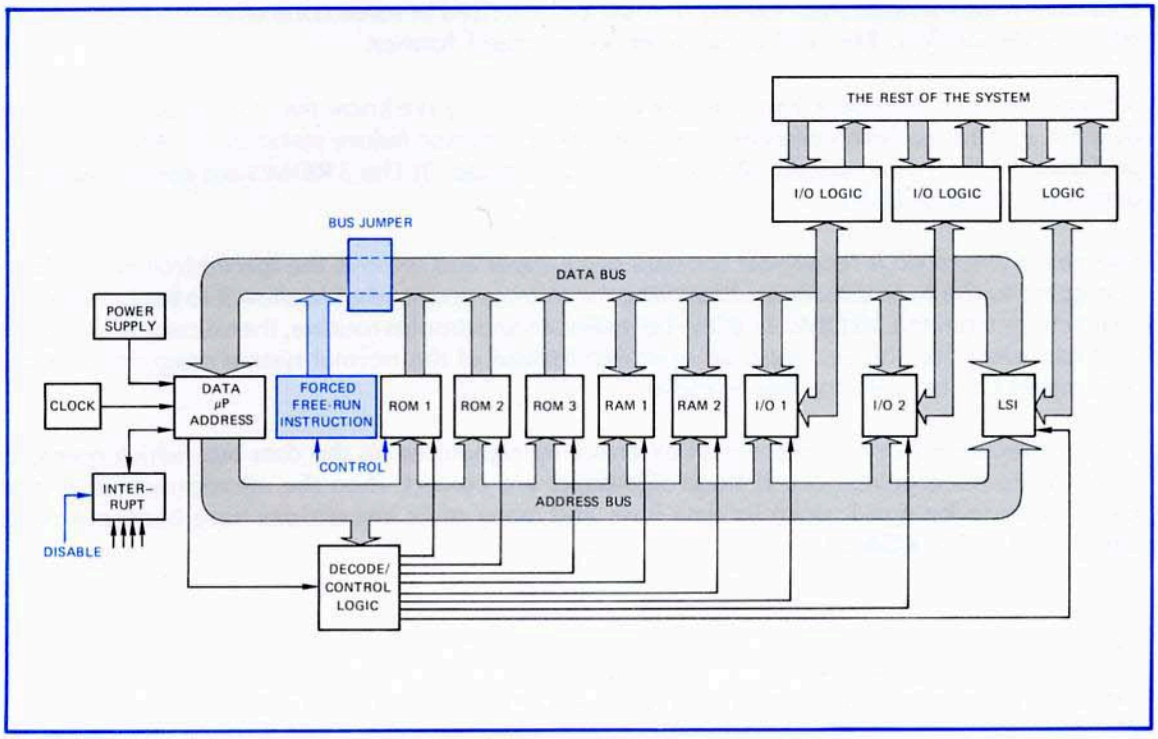


Figure 3. Application Example

The first thing we might do is get the system to free-run. Let's assume that there are provisions for breaking the bus between the microprocessor and the other bus elements. Then by removing all of them, (either mechanically or electrically) except the microprocessor, and then forcing-in some instruction that will cause it to repeatedly increment the address lines, (i.e., look for the next instruction) and disabling the interrupt control, the address lines will free-run continuously, scanning through the full address field.* At this point the microprocessor is running "open-loop." With the Start and Stop lines from the Signature Analyzer both connected to the most-significant address bit and triggering on the same edge, a measurement window will be defined as the entire address field. Connecting the Clock line of the Signature Analyzer to the microprocessor clock and selecting the edge at which the address is valid, the Signature Analyzer's input probe will then get sampled at least once at every address.**

By touching the probe to a logic 1 level (e.g., +5V line), a characteristic signature can be obtained. This is the result of shifting 1's in for every clock cycle that occurs while the gate (window) is open. This signature verifies that the Signature Analyzer's inputs are connected properly and that the microprocessor is free-running correctly. This is the characteristic "stuck at logic 1" signature for this test setup, and can be used to diagnose this condition elsewhere. A stuck at logic 0 is always a signature of "0000." The probe can now be used to verify correct signatures on each of the address lines by comparing them to signatures previously obtained from a known good circuit. Likewise, signatures can be checked on the address decode logic circuitry.

Assuming there are no bad signatures so far, ROM 1 can now be connected to the bus. It is still desirable to electrically open the data bus between ROM 1 and the microprocessor so that the outputs from the ROM will not force the microprocessor out of the free-run mode. Then, taking signatures on the data bus lines will verify all words of ROM 1. Adding ROM 2, (e.g., via chip enable pin or socket) will produce a new set of signatures on the data bus lines unique to the ROM 1, ROM 2 combination. Likewise, ROM 3 can be added. This process can be referred to as "expanding the kernel." Everything is still running open-loop with one portion being checked at a time and with minimum interaction between them.

Adding a RAM will not reveal much in the free-run mode, since it contains unpredictable data until it gets written into. Clearly, it must be exercised in some controlled manner to determine its disposition. This will occur after we proceed further.

It's time to take a step back and examine where we are: 1) We know the microprocessor can run through its address field correctly, and from micrprocessor failure statistics, its other functions probably work. 2) The address decoding circuitry is good. 3) The 3 ROM's are good, and 4) The data bus is clear of faults.

The next thing to do is reconnect the data bus jumper and remove the forced free-run instruction going to the microprocessor. Restarting the microprocessor should allow it to begin executing instructions from the 3 ROMs. If ROM 1 contains an SA stimulus routine, then a control line going to it can elect that this test loop be executed instead of the normal system program. Switches or jumpers can be used for this purpose.

Proper execution can be determined by checking signatures on the data bus (which now goes back to the microprocessor). If these signatures are correct, then the microprocessor is even more likely to be good, since its data lines and many of its instructions have been exercised, along with the 3 ROMs.

---

*The time for an entire address cycle in a 2 μs system with 2^{16} address states is only $\approx$ 130 ms.

**Multiple clock cycles may occur between each change in address depending on the microprocessor and the instruction that free-runs.

Now RAM 1 can be added. The SA program contained in ROM 1 executes a write/read test algorithm on the RAM, which in turn, generates a unique signature on each of the data bus lines. The addition of RAM 2 will generate a new set of signatures in a similar manner, verifying its correct operation.

I/O 1 and I/O 2 can be added, one at a time, and an exercise program can get them talking back and forth to the data bus, while signatures are taken. Looking at signatures on their external data output lines can verify their performance. And these signals can be followed into the I/O logic circuitry. Input paths to the I/O devices can be checked by supplying synchronous stimuli to them. One way to do this is to connect previously verified microprocessor controlled output latches to the I/O inputs via hardware jumpers or through software control. This permits controlled feedback on inputs normally out of the loop. In this manner much or all of the digital I/O interface circuitry can be tested.

Testing the LSI black box and related logic can be handled using similar techniques, in most cases. Apply whatever stimulus is required to exercise them and look at the signatures they produce.

The microprocessor itself can be put through an algorithmic exercise if further verification is desired; one that checks all the instructions used in its particular application and perhaps exercises its internal registers. Don't expect to be able to do an absolute 100% test on it, however. But if common failure modes can be identified, it may be useful to test these.

The interrupt system must also be tested synchronously. One way to do this is to enable one interrupt line, while disabling the others, which, via software configuration, would cause the microprocessor to jump into a unique test loop. SA can then verify that this has happened by producing the appropriate bus signatures.

Section IV
General Hardware Considerations

A preferred system structure is one that offers the ability to isolate components and prevent them from interacting with each other. Use of multiple P.C. boards, component sockets, switches, and jumpers are hardware techniques for doing this. In carefully executed SA applications, however, these mechanical tactics need only be used sparingly to obtain good testability. Software control can also be used, but it can never be as direct or reliable as hardware.

TEST MODE SELECTION

Specific techniques for getting into the free-running and other test modes are limited only by one's imagination. In the self-test or SA test mode it is necessary to get a portion of ROM program running that is not used during the normal operation of the product. Some methods for doing this are:

1. Plugging-in a test ROM in place of an existing one. In this case, the normal ROM, (in a socket) could be identified as good by the free-run test, and then replaced by the test ROM. Ideally, this ROM would be at the first executable memory address location interrogated by the microprocessor so that it will be the first to be exercised. The main disadvantage of this technique is that an available field test ROM is required in order to service the product, an I.C. socket is required, and the removal and insertion of ROM packages can cause them physical damage.
2. Use a service switch, (jumper or test pin) in the product or on a test fixture. This might be used to:
    a. Readdress a special portion of ROM containing the test program.
    b. Force an interrupt line defined in software to exercise the test program.
    c. Control the inputs to a 3-state buffer or input port on the data bus that the microprocessor can interrogate, or
    d. Disable a RAM in which the program attempts to store a "Ø", but (being disabled) a "1" gets read back to the microprocessor because of bus pull-ups.
3. Share switches used for other functions in the product. For example, the switches used for instrument address select on an interface communication bus (e.g., HP-IB) can be used to select various test loops if they can be read on to the data bus.
4. Merge a test ROM into the data bus via a jumper plug. For example: Bring the microprocessor data bus lines out to an edge connector on one side of a P.C. board, and the test ROM output lines to the other. A connector with shorts between upper and lower contacts can then close the path between the ROM and the data bus. It also provides a convenient service point for examining the data bus.

5. A self-test only program can be performed automatically when the product is powered-up or when a front panel switch (or unusual combination of switches), is selected.
6. If space permits, a P.C. board can be designed so that it can be pulled-out and reinserted into the same edge connector socket but from a different board edge, or the same edge upside down, causing the service test program to run.
7. Front panel switches used for normal product operation can be used for selecting test modes. Unfortunately, this assumes that a fair amount of the hardware is already working.

INPUT SIGNALS TO THE SIGNATURE ANALYZER

The Start, Stop, and Clock control signals used by the Signature Analyzer will now be discussed. A fundamental rule of any measuring instrument, such as the Signature Analyzer, is that it has negligible effect on the operating circuit to which it is attached. Although the inputs of the Signature Analyzer present relatively high resistance and low capacitance, circuits which have very low drive capabilities, (e.g., some low power CMOS and PMOS), or have a marginally high fanout condition present, may be affected by any additional loading. If this situation exists, it's a good practice to buffer the control lines going to the Signature Analzyer, (an inverter gate will do) to insure predictable operation. Standard test pins can be used for connection points but .030" diameter pins allow the Signature Analyzer's test leads to be connected directly.

The signals going to the Signature Analzyer must be reasonably clean at the clocking edge. Since the input speed characteristics of the Signature Analyzer will be as fast or faster than anything tested by it on the circuit, short noise spikes or glitches on the clock which may be ignored by the slower circuit components may trigger bad readings into the Signature Analyzer. Care must be exercised to recognize and control this noise to insure reliable performance. Also, insure that a good ground connection is made.

Two speed considerations are important for Signature Analysis. The first is the maximum clock input rate. The clock signal is always the fastest control input, and its maximum toggle rate is defined in the Signature Analyzer manual, (10 MHz for the 5004A). The second is the set-up time. This defines the minimum length of time during which data and Start/Stop signals must be present and stable before the selected clock edge occurs. This is an important consideration since circuit components that have long settling times may require much of a clock period to become stable, and not allow for very much Signature Analyzer set-up time. However, in conservatively designed microprocessor systems running with 10 MHz or slower clocks, sufficient set-up time is rarely a concern. Figure 4 illustrates set-up timing.

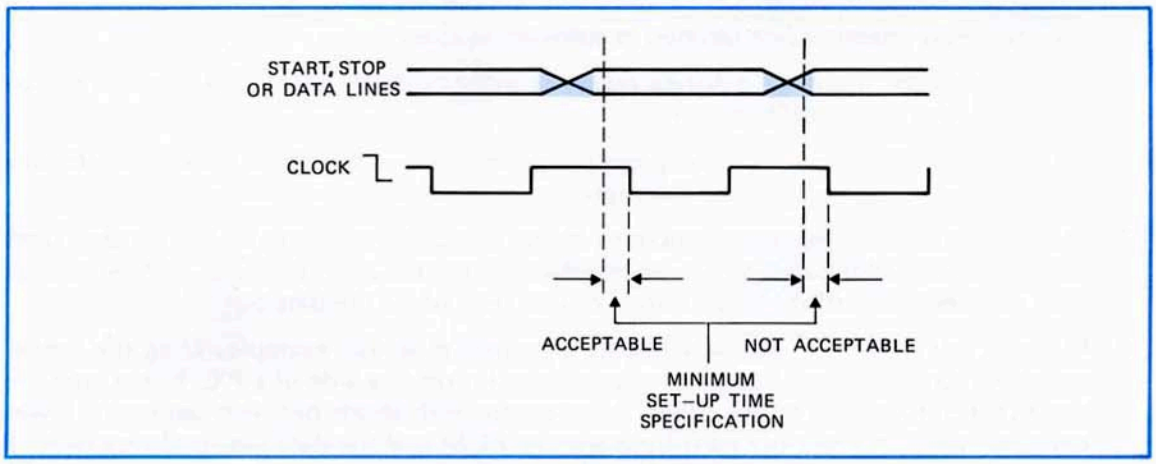


Figure 4. Set-up Time

If interrupts, memory refresh, or other asynchronous occurrences are integral to the circuitry being tested and cannot be eliminated in a test mode, hardware can often be included in the circuitry that will generate a special "valid data clock" signal, (just as a "valid address" signal is provided by most microprocessors which can be gated with the clock). The Signature Analyzer then would only sample data when it was stable and valid. Gating the clock off during other time periods when data input is not desired can also be done. Often, the gated signals required are already available.

A single clock edge selection in the Signature Analyzer is usually adequate for testing most of the synchronous nodes in a system. In some cases the clock edge must be changed to test certain system control lines, (e.g., read/write, interrupt enable, wait) which may not be in phase with the rest of the system.

The Start and Stop inputs, which define the measurement window, open and close a data sample gate in the Signature Analyzer. These are level controlled inputs that are synchronous to the clock. The same set-up time considerations used for the data probe apply to these lines, although the specs may be different.

Care must be taken to insure that data streams sampled within the window are repetitive in order to produce consistent signatures in each Start/Stop interval. In general it is best to get the Start and Stop signals as close to the microprocessor or controller as possible to reduce the chances of a hardware fault propagating into the Start/Stop control inputs. Unused high-order address lines on the microprocessor can be an ideal source. Additionally, the Start and Stop lines can be tied together allowing a convenient single connection to the circuit under test for the Start and Stop inputs.

Typical Start and Stop signal sources are address bits, address decoders, chip enable lines, interrupt acknowledge lines, and software controlled flags, data latches, and output I/O ports. One must be especially cautious when using chip enable lines that have clock data encoded on them for Start and Stop signals. If they are not in phase with the Signature Analyzer clock input edge, (as is often the case) you may find that the Start/Stop window is only **one** clock cycle long, (a legitimate condition for the Signature Analyzer, but not very useful for trouble-shooting). However, chip enables with clock data encoded on them can be useful as Clock inputs to the Signature Analyzer.

To generate a particular measurement window it is sometimes necessary to add extra hardware. Although this practice can usually be avoided by careful address field mapping, the addition of decoders, ring counters, and other hardware is sometimes practical for identifying specific Start/Stop windows. One novel technique for generating a Start/Stop signal is to decode a software "write" to a ROM address.

FREE-RUNNING

The free-run mode can be obtained by a number of different techniques. In a microprocessor system the normal data bus activity must be cut off from entering the microprocessor and be replaced by a forced instruction that will allow it to free-run. The most dependable method to open the bus is mechanical. Using switches (e.g., dip switch pack), jumper wires soldered on to a dip header or jumper packs such as the AMP 435704-8 plugged into an I.C. socket, bus buffers in an I.C. socket, or bus driving components in sockets are positive isolation techniques. Circuit hardware techniques, such as overriding chip enable lines, relies on an increased level of initial hardware confidence. In either case, a stuck or faulty bus line or control can prevent the μp from free-running. This condition can usually be detected using the "logic probe" function in the Signature Analyzer data probe. (Switch, jumper, and socket reliability in hostile operating environments should be considered before choosing to use them.)

With the data bus out of the picture, an instruction must be supplied to the proper μp data or instruction inputs that will cause it to continuously scan through the address field. Since these input lines are essentially floating, resistor pull-ups (or pull-downs) can be added to effect a suitable free-run instruction. There are several other options available as to how one might choose to accomplish this:

Using all pull-ups on the bus lines:

1. On some μps, this alone will produce a suitable free-run instruction.
2. Pull down on required lines via jumpers, switches, or
3. Use a single switch or jumper through diodes to do this.

It may be desirable to leave these resistors on the bus permanently, to insure the logic definition of a floating bus, and decrease bus settling time. Figure 5 illustrates how a "Clear B" instruction can be forced into an M6800 μp to accomplish free-running. Clear B was chosen to minimize the number of diodes needed. If an inverted bus or resistor pull-downs are used, the NOP instruction, (01) can be used to produce free-running with only a single bus line connection.

In processors where the instruction bus is either buried in one of the chips, (e.g., F-8) or shared with the address bus, (e.g., SCMP, 8048), other means can be used to accomplish free-running. For example, the dedicated ROM portion of the μp kernel can contain a free-run program loop, (and/or self-test and SA programs as well). But if ROM is already part of the μp chip, free-running may provide no real benefit. It might be best to just write self-test and SA programs in it. If these don't run, then the fault must be in this "kernel." Program selection can be accomplished by an interrupt or data input channel to the μp.

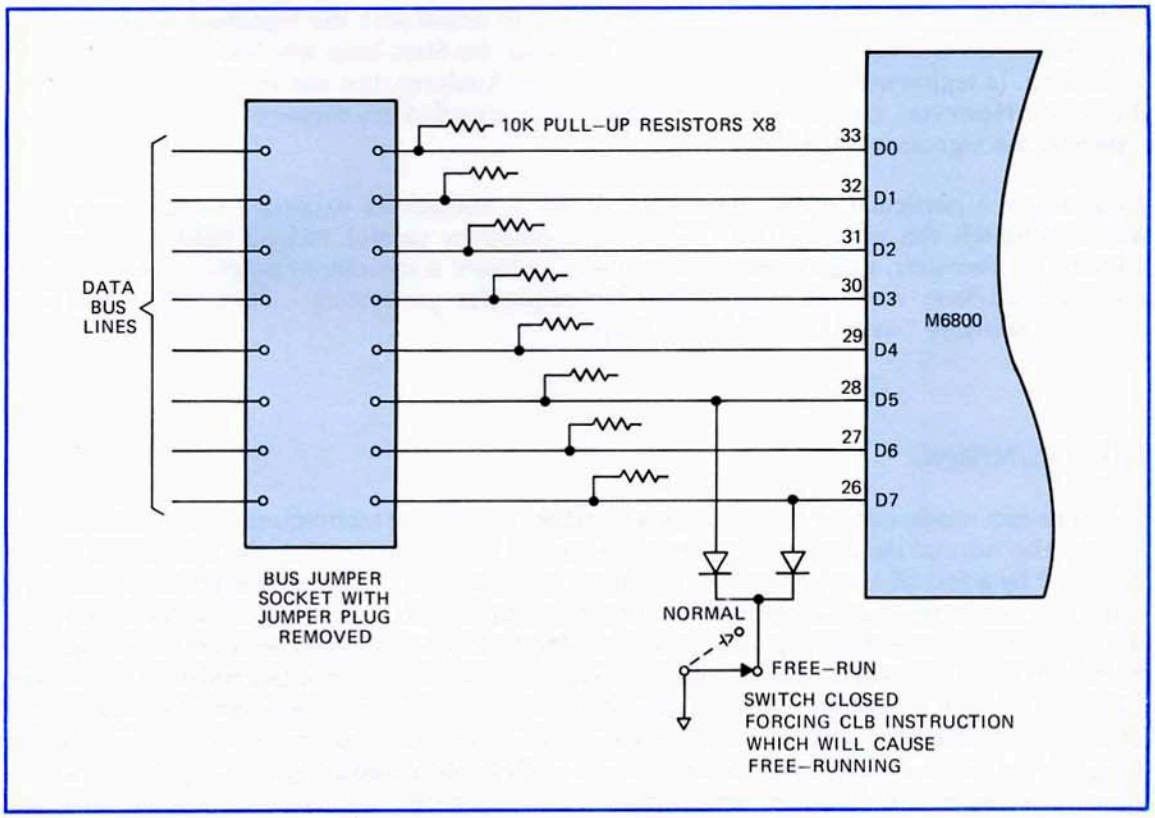


Figure 5. Forcing the M6800 to Free-run Using the Clear B Instruction and Removing a Bus Jumper

One should be careful about allowing a μp to free-run by executing a convenient but undefined instruction code. Its operation is not guaranteed and is probably not tested. What's more, another vendor's part may not do the same thing. Also there's the possibility that a μp manufacturer may later choose to define some instruction for that code which could prevent upward compatibility of the new part in your circuit.

Free-running is relatively easy to design into a μp system. This is also generally true for other types of systems that use some sort of counter or incrementer to direct system activity. If a counter doesn't exist, (as in some algorithmic state machines), it is sometimes practical to "wire-in" external counters, contained on a service board, to replace or augment the standard circuitry and provide a simulated free-run stimulus. (See section on ASMs.)

Section V
Techniques and Strategies for Designing with SA

This section will describe or deal further with the following topics and apply them to various hardware situations: Generating the stimulus to wiggle the nodes, software strategies, fault isolation techniques, looping on failures, use of the self-test feature, the time window concept, gating the clock, use of other test instruments, and designs to be avoided.

BUS STRUCTURES

Much of the power and flexibility of μp and ASM machines comes from utilization of the bus structure. In a typical application, the bus organization appears straightforward from the hardware point of view; but, with the system powered-up and running, data flow on the bus becomes a complex web. Logic analyzers in the hands of experienced operators can provide information as to what's happening in a running machine, and be invaluable as a development aid. However, unless they are designed for service repair or production testing environments, their use there may not be practical. Also, they lack the ability to compress long bit streams into smaller "signatures."

The troubleshooting process involves the detection of a circuit failure **and** localization of the fault(s) to the source. Finding the fault source in a bus system, where there is complex feedback, can be extremely difficult. But, opening the feedback paths and providing controlled stimulus to specific devices, enables thorough functional testing of circuits running "open-loop." This is analogous to making a freeway into a one lane, one way street with no on-ramps.

Theoretically, one could detect and replace faults in this manner and then tie the system back together (i.e., reclose the loops). This should insure the overall circuit to be good since, "If the sum of its parts is good, so is the whole." However, experience shows that this is not necessarily true. Once the feedback paths are reconnected and the test jumpers and switches are restored to normal, new problems may appear: drive deficiencies, interconnect problems, or timing incompatibilities. Although these classes of faults are much less common than hard failures, they compensate by being harder to detect.

A good way to avoid running into this sort of situation is to "expand the kernel." That is, build the system up one stage at a time by a verify and add process. Using the SA technique, the procedure would go something like this:

1. Test A (the kernel)
2. Use A to test B (another circuit)
3. Test A and B together
4. Use A and B to test C
5. Test A, B, and C together
6. etc.

Therefore, if step 4 showed C was running all right open-loop but step 5 failed, it is likely that an interfacing fault between C and the rest of the prior tested circuits (A and B) exists.

In some cases it may be practical to eliminate steps 2 and 4 since they imply greater ability to isolate components from one another, but at the expense of isolating hardware (sockets, switches, and jumpers). For example, suppose B and C are each ROMs on the data bus. Step 2 and 4 would assume that the bus could be opened between A and B and also between B and C.

It is important to have some means of opening or floating the bus to assist fault isolation and to allow for free-running. Some techniques for doing this are using software or hardware controlled device chip select lines, jumper wires, shorting plugs, and socketed components. Often, sockets can be designed into P.C. boards for the bus components. These can be eliminated at some time in the future if their added benefit doesn't justify their added cost, whereas it may be impossible to do the reverse without major P.C. board changes.

Although SA makes large boards more practical in terms of being able to do field repair on them, there are advantages to be had from the more expensive multiboard system structure; especially where the buses are concerned. Partitioning buses into multiple plug-in boards facilitates the bus fault isolation process. The free-running kernel can be enlarged by adding boards, until the bad one is found, (via bad signatures on a bus or a system crash). Additionally, a special extender board with a switch on each data and address bus line could be used between a bad board and the rest of the system. By closing one switch at a time, and observing the signatures, a specific bad bus line can be identified. With all data bus switches open, (but address switches closed), signatures on a known bad board can be taken to localize the failure, without the risk of feeding back bad data from its outputs to the rest of the system, which could invalidate the controlled test stimulus pattern.

The Signature Analyzer accepts data as ones and zeroes. For this reason, a node must always be in a predictable, repeatable, and stable logic state, ("Ø", "1", or in between) whenever the Signature Analyzer inputs data from it. Pull-up resistors, default bus drivers (bus drivers that pull the bus to a logic state when no other bus drivers are on), or the Signature Analyzer input resistance can be used for assuring this. One could also design the bus system such that for all address space utilized in the machine, some bus drive device is enabled. But, provision must be allowed for the free-run mode, where "all" possible address space may get addressed.

In most μp system buses, pull-up resistors in the 5—20KΩ range will suffice. The resistor values must be low enough to overdrive bus leakage and to charge bus capacitance fast enough to comply with the required Signature Analyzer's set-up time. They must be large enough in value, so that the weakest bus drivers can handle them in addition to their normal bus load. Therefore, resistor pull-ups may not always be suitable, due to a combination of high-speed or large bus capacitance (such as might be present in large multiboard bus interconnect structures with many bus elements). Consider a bus line with 10K pull-up and 100pf of line capacitance. An RC time constant of 1 μs results! Excessive bus settling time is one of the most common causes of unstable signature readings.

A default bus driver can be added that will actively pull the bus (one way or the other) when no other bus drive is enabled. Decode hardware using address and bus control inputs could supply the control. Allowance can be made for the free-run mode, where additional address states get generated. But what might be more difficult to account for is defined but unoccupied address field, (e.g., optional or future plug-in memory or I/O boards).

Another technique is clock gating (or clock qualify). There are instances where a bus must float or has data on it that, in terms of the μp, is random information (as might occur in a DMA operation or an asynchronous input setup). Inhibiting the Clock input to the Signature Analyzer during this period will prevent unwanted data from getting into it. Sometimes the Start/Stop window

can be set-up to exclude only the unwanted data without losing desirable information. Both of these later techniques usually require additional software controlled outputs or decode hardware in the circuit or on a test fixture, and may prescribe multiple Start/Stop set-ups for the same node.

If the data bus write-enable signal is gated with the Clock input to the Signature Analyzer, data will only be input when the μP, (or the device in control of the bus) is outputing valid data. This virtually eliminates the possibility of inputing unstable data from a floating bus. Two consequences of using this technique are:

1. the write-gated clock cannot be used during free-run because writes never occur, and
2. data present during a read cycle will not be input to the Signature Analyzer. To test this data, the μP can echo (re-transmit) it by means of a subsequent write cycle. Also, it need not actually be written into an existing bus device (i.e., an undefined address can be used).

When it is not practical to isolate bus components from each other, other techniques can be used to find a fault. For example, if any two bus lines with activity on them produce the same signatures, there is a good chance they are shorted together. A characteristic "1" or "Ø" ("ØØØØ") signature on a line with no activity says it's probably stuck or shorted to V_{CC} or ground. An HP 547A Current Tracer used with an HP 546A Logic Pulser can be handy for finding the source of the problem by injecting current and following it to the fault. By forcing the chip enable line of a bus device to the off state, a Current Tracer can detect any current activity on its "disabled" bus pins which would indicate improper operation (e.g., stuck output, faulty disable input).

Using one of the multiple chip enable pins on bus components with internal decoding logic, as in many 6800 family parts, can be useful for isolating a device that could be coming on the bus at the wrong time because of a fault in its internal decoder. In a system with many such parts one should weigh the value of using these buried address decoders against the added impact they may have on fault isolation.

A good stimulus for bus lines is a simple binary incrementation. This is accomplished automatically for the address bus when the microprocessor is free-running, and can be generated by a simple algorithm for the data bus. But, just addressing words of ROM on the bus will likely wiggle the data bus lines enough to check them out. Other techniques will be discussed for specific bus components in the following sections.

ROMs

The basic principle for testing ROM's is to apply all possible input codes while checking all the outputs. There are several ways of doing this in addition to free-running. In a product resident "self-test" routine, ROMs are read, as data, into the microprocessor while a data compressing software algorithm is performed, which leaves a characteristic residue in an accumulator or register. The correct residue insures, to a predictable level of confidence, the validity of the ROMs. (Cyclical redundancy and checksum are two of many algorithms that can be used for this.) Be careful, however, to define any normally unused or "don't care" ROM states, which will, for most self-test and SA test loops, be included in the ROM verification program.

A ROM test program can be included as part of a larger product test sequence or be an autonomous program. It can be designed to contribute to a GO-NO/GO condition for the overall product or provide specific failure information, such as "there has been a ROM failure," or even more explicitly, "ROM #3 has failed." Self-test failure information can be via existing front panel display elements, additional indicators, or LEDs on PC boards. Error codes and messages can even be used. Be cautioned, however, that the less hardware assumed good beforehand, the better. For this reason, consider negative fault indicators, (i.e., a **good** test results in an **on** indicator). This way, a faulty indicator will always show a failure condition.

It's always nice to be able to remove or disable each ROM individually. But, when this luxury cannot be accommodated, other techniques can be used to isolate one ROM from another and from the other bus elements. By running in the free-run mode and verifying proper address decoding to each of the bus elements, (including the ROMs) one can feel confident that no more than one bus device is enabled at a time, (assuming their output enables all work). While still in the free-run mode, a technique referred to as "address window fault isolation" can be used. This involves connecting the Start and Stop Signature Analyzer input connections to correspond with the address decode field associated with a selected ROM. The Signature Analyzer's data probe can then check each line of the data bus for the proper signature, and since it inputs data only during a field of addresses associated with that ROM, a bad signature implies a bad ROM, (that is, unless another device is on the bus when it shouldn't be). Sometimes the same address decode circuitry used for a ROM can also control the Start and Stop inputs of the Signature Analyzer. By moving the Start/Stop address window, individual ROMs or groups of them can be verified. Figure 6 illustrates this technique.

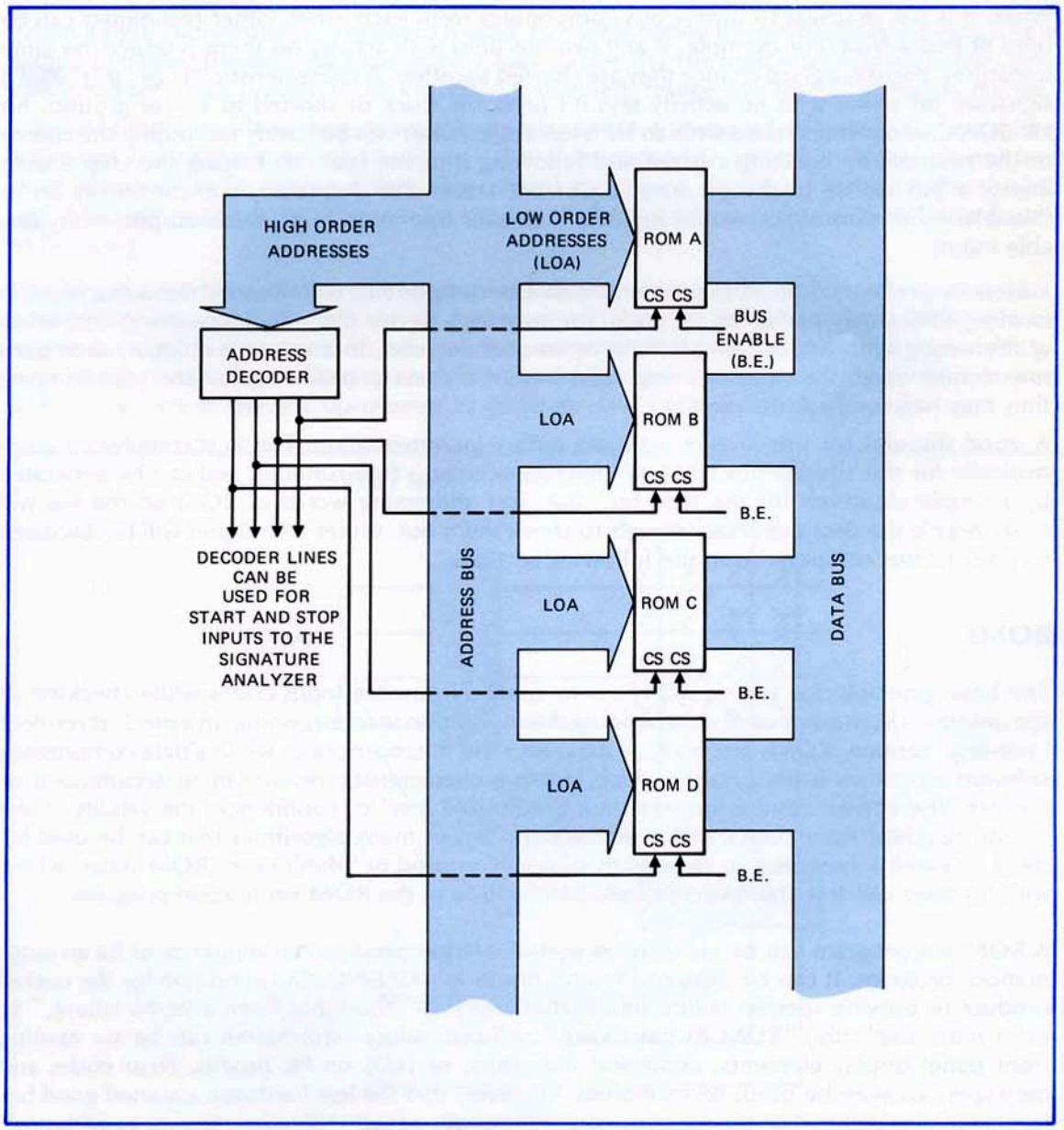


Figure 6. Getting START/STOP Signals from Address Decoders

Without the ability to physically remove all other bus elements, there is a possibility that a bad signature could result from one of the other bus components, (e.g., through a faulty on-chip output control circuit). Techniques using a Current Tracer, (such as the HP 547A) can be useful for dealing with these situations by following the current path to a stuck bus output pin on an I.C.

A more powerful way to deal with ROMs is by running a special SA test program under microprocessor control. A cyclical redundancy, checksum, or other test can be run on each ROM during an SA test loop or loops. In addition, by generating a test program which interacts with the hardware (in this case ROMs) being tested, the program itself can provide fault information. If the test program loops back to its beginning when a fault occurs, for example, its execution time in clock cycles will be a function of how far it was able to progress before finding a failure. By connecting the Start and Stop inputs to a line that changes every time the test loop completes a loop cycle, (via a high order address line, controlled I/O, et. al.), a variable length measurement window will be created. So, for a failure occurring early in the test loop, a lesser number of clocks will be input during a shorter measurement window than for a failure that occurs later in the loop. If the Signature Analyzer's data probe is connected to a constant logic "1" level (e.g., Vcc), a continuous stream of "1 s" are shifted in throughout each pass of the test loop. By cataloging the various signatures obtained for all possible loop (component test) lengths, a "fault dictionary" can be obtained, from which the first component to fail can be determined. Figure 7 is one possible flowchart for this procedure.

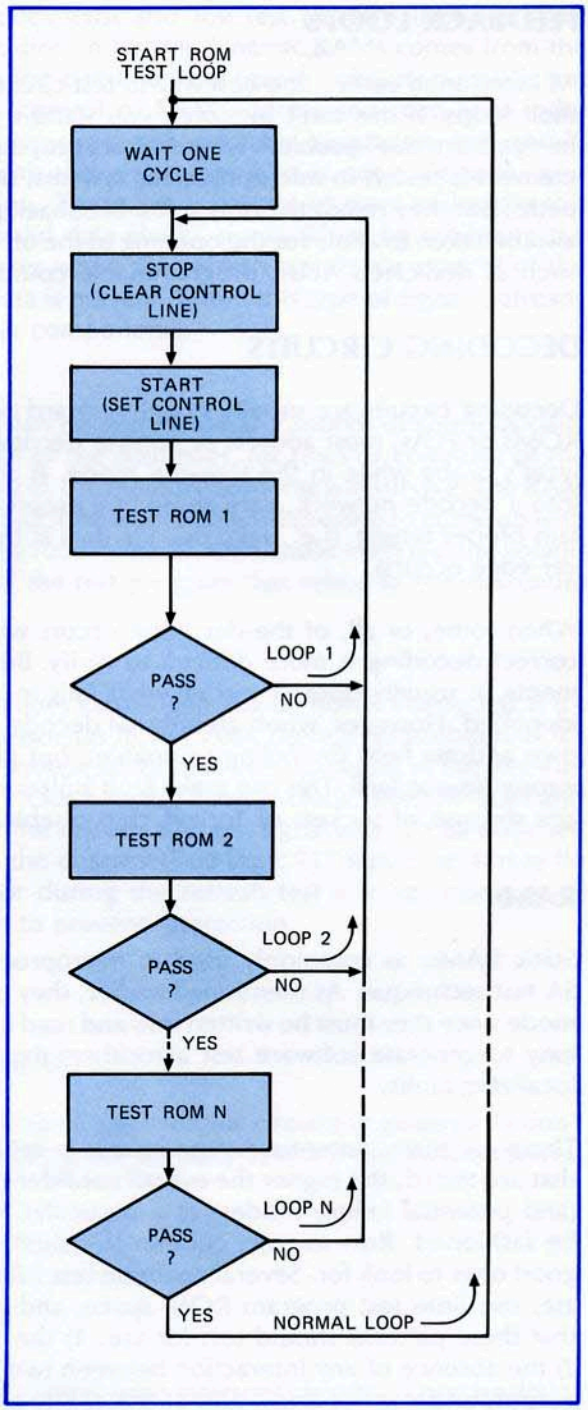


Figure 7. Loop on Fail Flowchart. SA test cycle length varies with first fault location to produce a characteristic Vcc signature.

This is a very powerful technique that can pinpoint device faults within a large group of components with only one input setup to the Signature Analyzer. It does, however, indicate only the first failure that it finds. Additional faults would not be detected until this first one was repaired. It also assumes that a certain amount of the hardware is good to begin with, to allow the test program to run.

It may be desirable to select smaller loops which can test specific ROMs or groups of ROMs, (or other components) to pinpoint multiple faults. Other variations of this technique can be used to provide less or more diagnostic information (e.g., identify a bad board or module, or a specific ROM output pin, respectively).

FEEDBACK LOOPS

As mentioned earlier, the best way to test circuits with either signal or data feedback is to open their loops. If this can't be done, you stand a good change of running into the old "garbage in—garbage out" problem where errors propagate around the loop and you never know which element is faulty. In microprocessor systems, data buses are the most common feedback loop paths, but they're not the only ones. Feedback paths in the peripheral circuitry dictate that care also be taken to allow for the opening of the off-bus or secondary-bus feedback loops in circuits such as dedicated ALUs, programmable counters, and data converters.

DECODING CIRCUITS

Decoding circuits are usually straightforward combinatorial logic. Using gates, decoder chips, ROMs or PLAs, most address or control decode lines can be checked with the Signature Analyzer's probe while in the free-run mode. If address or "data valid" control signals are gated into a decode network, care should be excerised that the Data input and Clock signals maintain proper timing, (i.e., valid decode data at the probe should be set-up before the clock trigger edge occurs).

When some, or all, of the decoding occurs within an I.C. (as in several 6800 family circuits), correct decoding is more difficult to verify. But, in field service, or even production environments, it usually doesn't matter what fails in a component, as long as the bad part can be identified. However, when an internal decode does fail, it not only generates a failure over its own address field (by failing to enable), but perturbs other address fields by enabling at the wrong time as well. This can make fault isolation somewhat more challenging, and may encourage the use of sockets or forced chip disable test pins.

RAMs

Static RAMs, as commonly used in microprocessor systems, are generally well suited to the SA test technique. As mentioned earlier, they cannot be tested adequately during the free-run mode since they must be written into **and** read in a controlled manner. But, there are numerous, easy to generate software test algorithms that can do a respectable job of testing them and localizing faults.

There are many potential failure modes in modern high density RAMs and the more of these that are tested, the higher the overall confidence. If specific knowledge of the internal structure (and potential failure modes) of a particular RAM is available, an intelligent set of tests can be fashioned. Row-to-row, column-to-column, decoder, and lead bond failures are usually good ones to look for. Several common test patterns can be considered which are easy to generate, use little test program ROM space, and run quickly. The essential properties of a RAM that these patterns should test for are: 1) the ability to write and read a 1 and Ø in every cell, 2) the absence of any interaction between two memory cells, 3) the proper operation of input, output, and control lines, and 4) the ability to uniquely address every word. Some common patterns used to test these properties include: checkerboard, walking 1's and Ø's, storing the address (successive numbers), and storing the contents of the ROM in the RAM.

The RAM test can be part of a power-on or self-test program and also be used in the SA test program. Operating in a loop-on-fail program, (as discussed in the ROM section) the Signature Analyzer can be used to identify directly a specific bad RAM. And, the program loop can be set up to exercise anything from a particular RAM or group of RAM's to the whole system, just as was the case for the ROM test loop. The advantage of a RAM-only test loop is that one knows specifically what's being tested, and when a bad signature is found that it's a RAM that has failed.

In testing dynamic RAMs, similar failures modes exist and the test patterns used for static RAMs also apply. The only additional complication in testing dynamic RAMs comes from the refresh circuitry. Since the RAMs are tested in-the-circuit and at speed, volatility is usually not a problem. But, when refresh circuitry gains control of RAM address and read lines independently and pseudo-randomly from the microprocessor, unstable signatures may result. Even if the refresh circuitry is synchronous with the master clock, it is not necessarily synchronous with the test program that is running. There are a couple of ways to handle this situation. One involves the generation of a "valid data clock" signal. Thus, by inhibiting the Clock line signal going to the Signature Analyzer when the refresh circuit has control of the RAM, data is only sampled when valid RAM data is present. Often this type of signal is already available and being used for enabling other bus components.

By disabling the refresh circuitry altogether, it can be eliminated as a source of spurious data. If the RAM test loop program scans through all of the addresses, (or rows or columns) of the RAMs within the maximum allowed refresh period, the memory contents will survive. The RAM is refreshed by the program that is testing it. In fact, one would be hard pressed to come up with a good RAM test program that didn't meet these requirements. One could even test the volatility of a RAM by programming wait loops into the test program that relate to the maximum memory refresh cycle time.

As far as testing the refresh circuitry itself, techniques vary depending on how it's designed. If it's driven by and synchronized to the microprocessor (as in the Z80), there's no problem. If it's asynchronous, then you can often treat it as an independent machine operating in the free-run mode. By connecting the Signature Analyzer's Clock to the refresh clock, and the Start and Stop lines to the MSB, (most-significant-bit) of the refresh address, signatures can be obtained from the rest of the refresh circuitry as well as the characteristic logic "1" signature. It may be necessary to reset and halt the microprocessor during the refresh test if it has interrupt or cycle stealing capability over the refresh circuit to prevent interaction.

ASYNCHRONOUS CIRCUITS (IN GENERAL)

There are four basic ways to deal with the testing of asychronous circuits in general: 1) don't design them in, 2) make them synchronous during the SA test, 3) test them by themselves, and 4) don't test them.

INTERRUPTS

In order to obtain meaningful and repeatable signatures from interrupt circuitry, a precise timing relationship between the Clock, Start/Stop window and data input must exist. But, since interrupts are generally used with asynchronous circuitry, special provisions must often be made to obtain a good SA test environment. In general, an interrupt signal might occur at any point in a program and its precise time of occurrence cannot usually be controlled by the microprocessor.

There are many common types of circuits that might provide interrupt signals: keyboards, display printers, RAM refresh, DMA (direct memory access), communication, and an assortment of servo control and data acquisition circuits. Often multiple interrupts occur through vectored or priority networks.

It's best to look at interrupts as closed loop structures whose feedback loops must be opened in order to apply SA. A good place to open the loop is often at the microprocessor interrupt input. By opening the interrupt line at that point, a controlled test situation can result. An interrupt request may be sent, but it won't get into the microprocessor. With the SA test program running, a forced interrupt (via switch or jumper at the microprocessor) could cause a jump to an interrupt subroutine different from the one it would normally go to; one that would result in an exercising pattern designed to synchronously check-out the circuitry involved with generating that interrupt signal.

One way of doing this is to replace the ROM address space where an interrupt would normally go, with an SA test ROM. That is, the SA ROM would replace, either physically (via socket) or by address decoding (via switches or jumpers), the ROM instruction code normally found at a portion, (or page) of memory. See Figure 8a, and b. Another way would be to include a normal/SA decision test in the interrupt subroutine. See Figure 8c.

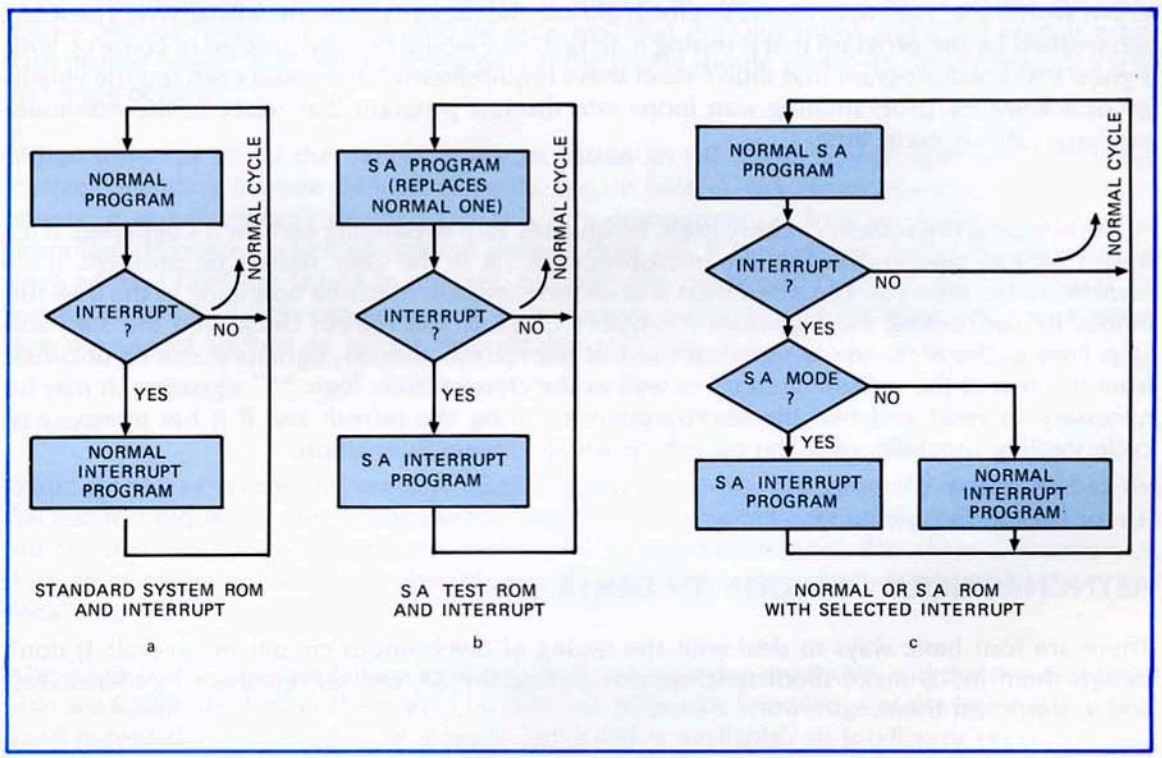


Figure 8. How an SA Interrupt can be Selected in Software

Any asynchronous circuitry involved in the interrupt path should be synchronized or bypassed. Feedback loops should be opened and tested separately. The system clock can usually be used for the Signature Analyzer's Clock input. Start and Stop signals can be from an interrupt acknowledge line, or software generated via an extra address line or a data bus controlled device, (such as a latch or output port). If multiple interrupts exist, the "interrupt system" can be built up from an enlarging the kernel approach. That is, all interrupts can be disabled but one. That one can be verified using SA and then the others can be enabled and verified in succession.

Closed loop verification can also be performed by using an SA test loop program that exercices the interrupts as they are normally connected, with asynchronous delays bypassed. This can identify failures, but specific fault location can be difficult because of the feedback paths involved. Another technique is to enable an interrupt before the Start/Stop window and then test the consequence of that interrupt.

ONE-SHOTS

Monostable multivibrators, (or one-shots) often provide practical solutions to some types of circuit design timing requirements. When subclock periods or long or adjustable time delays are required, they can result in a better design tradeoff than purely digital alternatives.

One-shots in an SA-compatible circuit must be able to produce synchronous, predictable, and repeatable pulses in order to provide consistent signatures. The leading edge of a one-shot can typically be triggered in a controlled manner by the microprocessor during an SA test program. The trailing edge, however, cannot. When a one-shot pulse width is more than a few clock cycles long, it gets increasingly difficult to maintain an accurate timing relationship between the one-shot and the system clock. Often the best course of action is to make provisions for bypassing the one-shot or forcing its pulse width to be narrow. This can often be accomplished by either removing the capacitive timing element in a one-shot or by replacing it with a smaller value. A jumper on the board can be a convenient means for doing either of these. An advantage of leaving enough capacitance connected to the one-shot to produce a pulse width of more than one-clock period is that the one-shot functional operation can be verified during the SA test loop. Functional operation of one-shots can also be detected with the pulse-catching "logic probe" feature of the Signature Analyzer.

Designs that require a long one-shot pulse width in order to allow circuits to be sufficiently exercised should also be avoided. Or provisions should be made to circumvent this situation. One way this might be done, for purposes of testing, is to jumper-out the one-shot output, and jumper-in a microprocessor controllable line, (such as an output latch) operating outside of the actual portion of the circuit under test. This way the one-shot can be "simulated" synchronously by another circuit. Another technique is to use the microprocessor to control the clear input of the one-shot.

DISPLAYS

Display circuitry can vary widely from one application to another. Some rely heavily on the microprocessor, and may represent a significant overhead, while others can be almost totally self-sufficient.

Displays have the unique advantage of providing a visual verification of their own operation. And some failures (e.g., missing segments) can be diagnosed directly. Often a power-on test program can be used to light-up all indicators and display segments for a few seconds, allowing a quick visual check to be made. As part of a self-test program, display characters can be shifted across a display at visually observable rates to test character generation capability. These techniques can assure the integrity of the display and indicator circuitry, but do not supply sufficient information to diagnose all possible faults which could cause a malfunction to occur. For this, an SA test exercise program can do the trick.

Displays are usually open-loop circuits. They have data input to them and then they output to indicators. In multiplexed displays, there may be internal feedback paths (e.g., around a shift register or counter). If the display circuit can be totally controlled by the microprocessor, then the task of applying SA to it is fairly straightforward. Just generate a suitable stimulus algorithm that wiggles the nodes, and take signatures. But, if the display circuit contains asynchronous circuitry, (e.g., one-shots, free-running display refresh, or microprocessor interrupt) additional steps must be taken to place it under microprocessor control, or test it independently of the microprocessor, (see Counter/Shift Register section.) Once a display has been verified to be good, it can then be used to help check out other circuits, (e.g., echo keyboard entries, display error codes).

In CRT raster scan display applications, the ability to lock (i.e., synchronize) the horizontal and vertical scanning circuits to the microprocessor will help facilitate a good SA application. In doing this, however, consider the effect that slowing down or speeding up the video might have on the CRT drive circuitry. It may be desirable to disconnect the display generator from the video line in order to avoid overloading the CRT drive circuitry. Restricting the use of one-shots is also advised (see One-Shots section). And, if the display circuitry is interrupt driven, uses DMA, or shares common memory with the microprocessor, appropriate overriding and synchronizing provisions must be made to insure that controlled signatures are generated.

KEYBOARDS

Keyboards, switches, and pushbutton circuits have the advantage of being able to provide their own stimulus via "finger" interface. The correct operation of a switch can often be observed directly by the response of the product it controls. If, for example, there is an operating display, it can be used to echo key switch closures.

When a static, non-scanned, keyboard is used, troubleshooting the keyboard circuitry can be as simple as tracing through the logic using a logic probe, (or the logic state feature of the Signature Analyzer's data probe). If the keyboard is scanned, testing can be quite another matter.

Scanned (multiplexed) keyboards usually employ either a keyboard encoder I.C. or a microprocessor controlled scanning circuit. In general, encoder I.C. interface circuits do not lend themselves well to SA testing. Often, key control and timing signals are buried within the I.C. and adequate scan synchronization cannot be attained. Each application must be evaluated individually.

On the other hand, when the microprocessor is used to scan a keyboard, a controlled and SA-controllable situation can exist. An SA key scan test loop, (likely to bear major resemblance to the normal key scan routine) can generate outputs to the keyboard matrix on which signatures can be taken. Pushing appropriate keys will close the loop and cause the matrixed lines going back to the microprocessor to generate new signatures. See Figure 9.

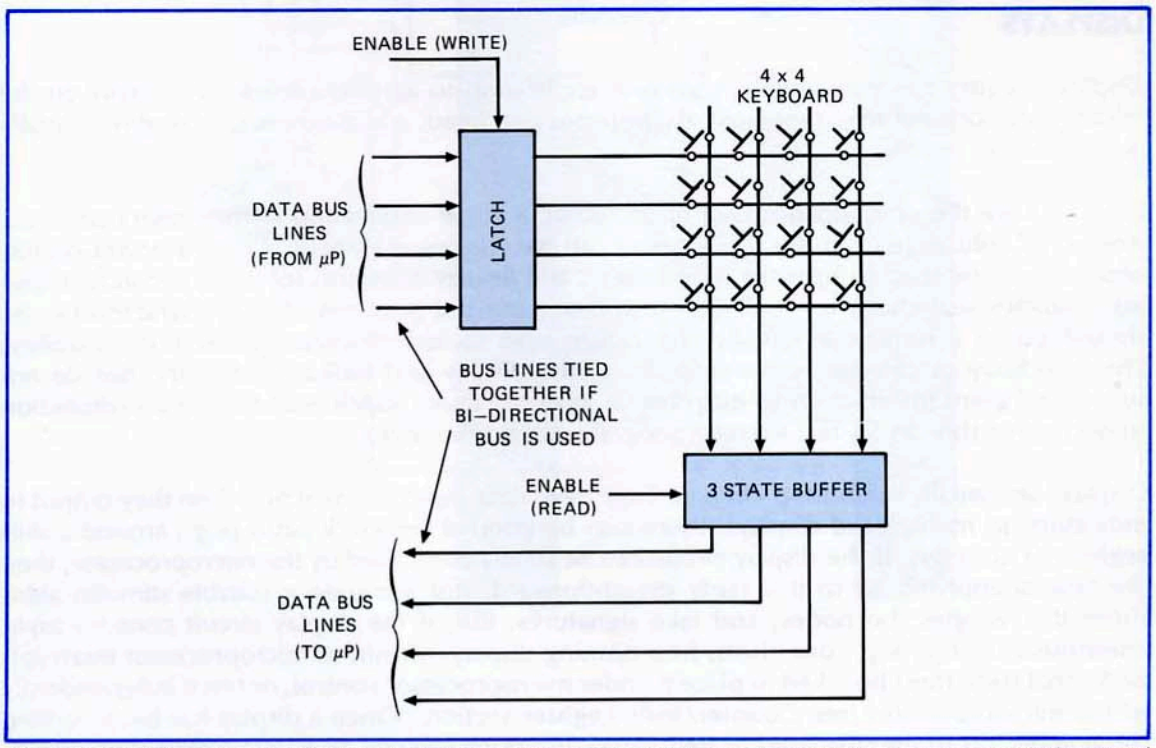


Figure 9. Typical μP Scanned Keyboard. Signatures taken at the latch output and buffer input can verify key switch performance.

If the keyboard operates by interrupting the microprocessor when a key is pushed, the techniques discussed for dealing with interrupts apply. Of course the interrupt can be inhibited simply by not pushing a key. But, since keys must be pushed in the course of checking out a keyboard, it can often be useful to be able to disconnect or control the keyboard interrupt line going to the microprocessor.

One might also view a keyboard as an input device which can be emulated by microprocessor controlled outputs. Using jumpers or connectors, (extra edge connector space on a keyboard P.C. board can provide a "free" connector) to route known good microprocessor-controlled SA output stimulus to keyboard circuitry that gets fed back into the microprocessor, can check out much of a keyboard interface without ever touching a key switch.

ANALOG CIRCUITS

Analog circuits are not generally testable with SA. SA is a digital technique, but it can be used for limited and some indirect testing of analog/linear circuits. For example, sometimes an analog circuit can be made to look digital, (e.g., program a DAC to logic 1 or Ø voltage levels and take signatures on its output). Or, by injecting controlled analog inputs, (e.g., a pulse generator or power supply) into an analog circuitry, such as an ADC, which interfaces to digital circuits, correct operation can be verified by taking signatures on the digital portion. Conversely, a previously verified digital circuit can be used to exercise an analog one. The resulting analog signal responses can then be measured with conventional test equipment (e.g., scope, DVM, counter), or in some cases, fed back to the digital portion thus "closing the loop." Care must be exercised when dealing with analog circuitry that the speed and timing requirements for the Signature Analyzer are met. For example, when probing analog nodes, a slower clock rate input to the Signature Analyzer may be necessary in order to allow the slower analog circuits to stabilize before getting input.

MULTIPROCESSORS

In multiprocessor products, one processor is usually capable of controlling, or at least communicating with the others. This primary "control" microprocessor is usually the only one that needs an SA ROM program and the ability to free-run. After verifying its own operation, it can then exercise the other microprocessors and related peripheral circuitry.

If there is no effective control path from the primary microprocessor to a secondary one, it also follows that there is not too much direct "interaction" between the two. If this is the case, it may be desirable to break or inhibit whatever communication path there is between them, (could be done in software) and test the secondary microprocessor and related peripheral circuitry by itself; getting it to free-run and execute its own SA test program. It may be important to insure that the particular microprocessor being tested is the only one that has access to any common hardware (e.g., RAM) during the Start/Stop window. This will prevent signatures that are contingent on circuits outside the desired test field, and thus direct the fault isolation to a smaller region.

COUNTERS AND SHIFT REGISTERS

Counters and shift registers under microprocessor control can be exercised and tested in much the same manner as other microprocessor controlled components. For example, consider the circuit in Figure 10 composed of a series of 7490 type decade counters.

If the reset and input controls are, or can be, placed under microprocessor control with the signal normally feeding the counter input line replaced by one generated from the microprocessor, then an SA test loop can be fashioned to exercise the counters and generate controlled signatures.

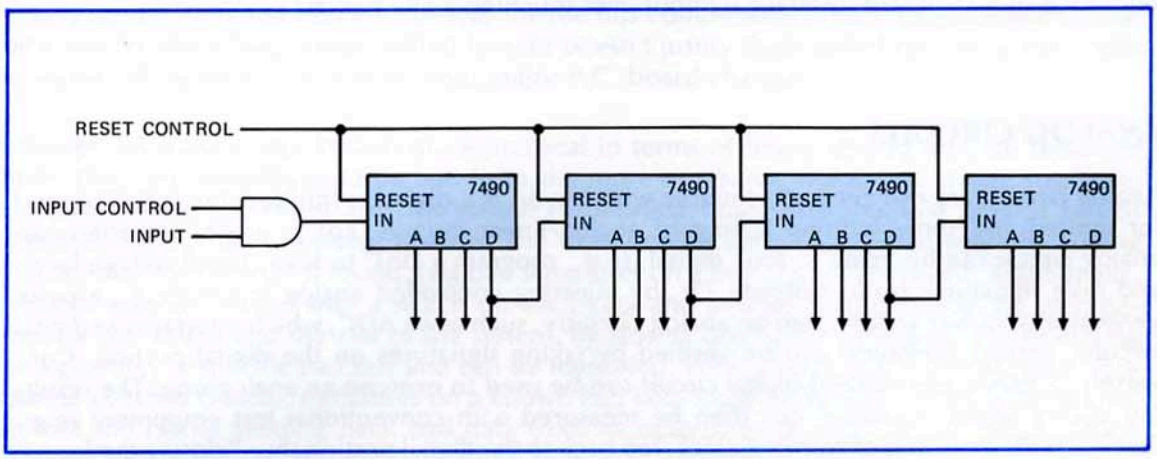


Figure 10. 7490 Counter Chain. Since each of the 7490s functions identically, identical signatures should be produced at each stage's output pins when the Signature Analyzer's inputs are connected to the I.C. under test.

Another way is to test them independently. This can be done by first ensuring that the reset control is either held off (not reset) or tied to a synchronous (with respect to the counters) line which would predictably reset the counters when a count state in excess of their overflow is reached. Then, by supplying an input pulse train below the maximum allowed rate for the clock input of the Signature Analyzer and for the counters, correct operation of the counters can be verified using SA. An effective and easy way to take signatures on a chain of identical counters is by using an I.C. test clip (such as those made by AP, Pomona, et. al.) with the Signature Analyzer's input signals connected to it. Plug the Clock input connector on to the clip's pin going to the input of the counter I.C., the Start and Stop to the most-significant-bit output pin, or carry out, and the data probe to the various outputs and V_{CC}. Touching V_{CC} checks the Start/Stop window length and verifies that the I.C. is dividing properly. The output signatures test the individual stages.

A counter I.C. may count properly internally, but still have one or more bad outputs. This is because connections internal to the I.C. are often used to propagate signals from one stage to another, and buffers and bonding leads (which can fail) are used to bring these signals to the external output pins. By moving the I.C. clip from one counter I.C. to another, all of them can be checked against a common set of signatures.

If a 7490 device is connected as illustrated in Figure 11, with Start, Stop, and Clock connected as shown, the indicated signatures will result. Duplicating this set-up can be a simple means of gaining familiarity with the Signature Analyzer.

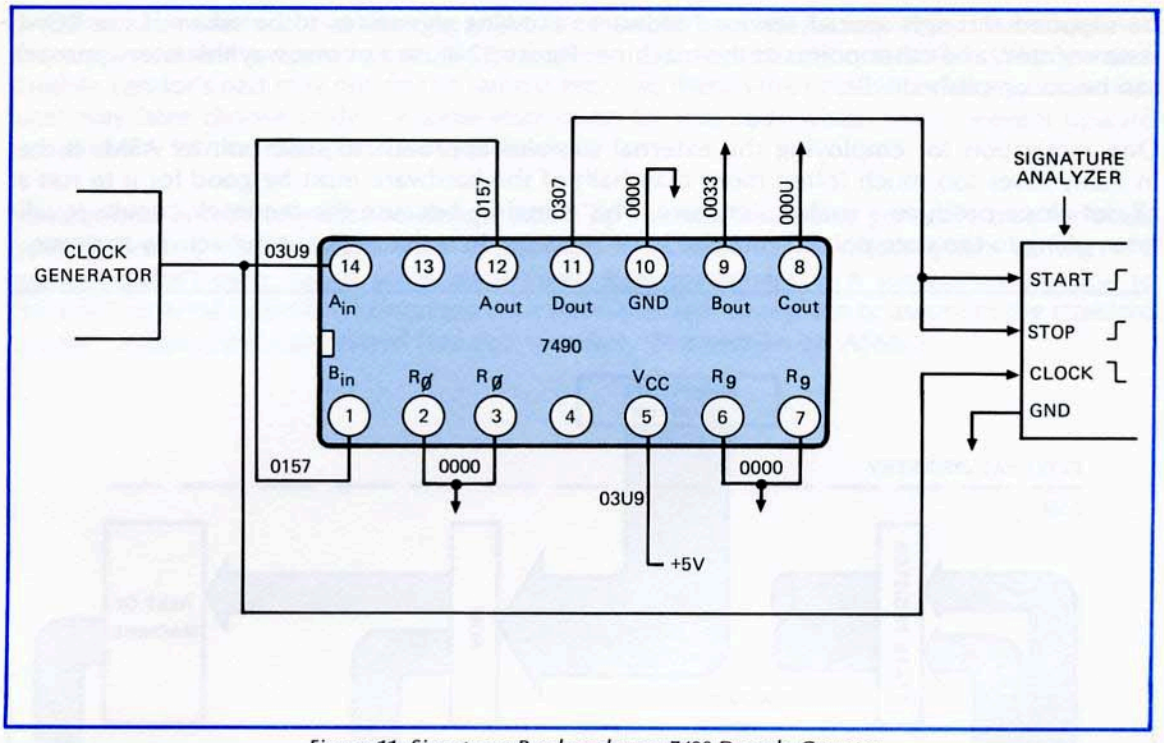


Figure 11. Signatures Produced on a 7490 Decade Counter

When testing long ripple counter chains, timing skews can add up to long propagation delays. If these are not accounted for, unstable signatures can result due to delay uncertainty and jitter. In fact, this phenomenon, if not controlled, can cause signatures to vary from one "identical" product to another. The best approach is to make sure the period of the clock input going to the ripple counter chain, (and to the Signature Analyzer) is no longer than the worst case maximum propagation delay of the entire chain.

The same techniques for dealing with counter chains can often be applied to shift register chains. Additionally, however, there must be a controlled input data stream provided to them to get the shift stages to toggle, so that both logic 1 and Ø shifts can be checked. If counter or shift register chains are synchronous, propagation delays are seldom a problem.

ASMs

Algorithmic State Machine (ASM) structures, comprised of state registers, ROMs, and feedback logic, can be good candidates for the SA test technique. Although not as flexible as microprocessor systems, a carefully designed ASM can benefit greatly from SA. The same guidelines and principles used for μps apply to ASMs as well: building the kernel (state pointer, ROMs, then logic), free-running, opening feedback loops, and generating SA stimulus test routines.

If the ASM state register has counting (or incrementing) capability, it is generally not difficult to get it to free-run. Register presetting control lines used for branching can be disabled, which then would allow the state counter register to increment continuously. However, if it's implemented with non-counting elements and acting as a state "pointer" (as opposed to a state "counter"), then free-running may not be possible. Instead, free-running can either be abandoned, at the expense of reduced fault isolation capability, or external free-run stimulus can

be supplied through special service hardware, allowing signatures to be taken at the ROM, state register, and other points in the machine. Figure 12 illustrates one way this later approach can be accomplished.

One motivation for employing the external stimulus approach to state pointer ASMs is that in many cases too much (often more than half) of the hardware must be good for it to run at all, let alone produce a usable signature. This is mainly because the feedback circuits (qualifiers) going to the state pointer must be left connected in order to get useful activity to occur.

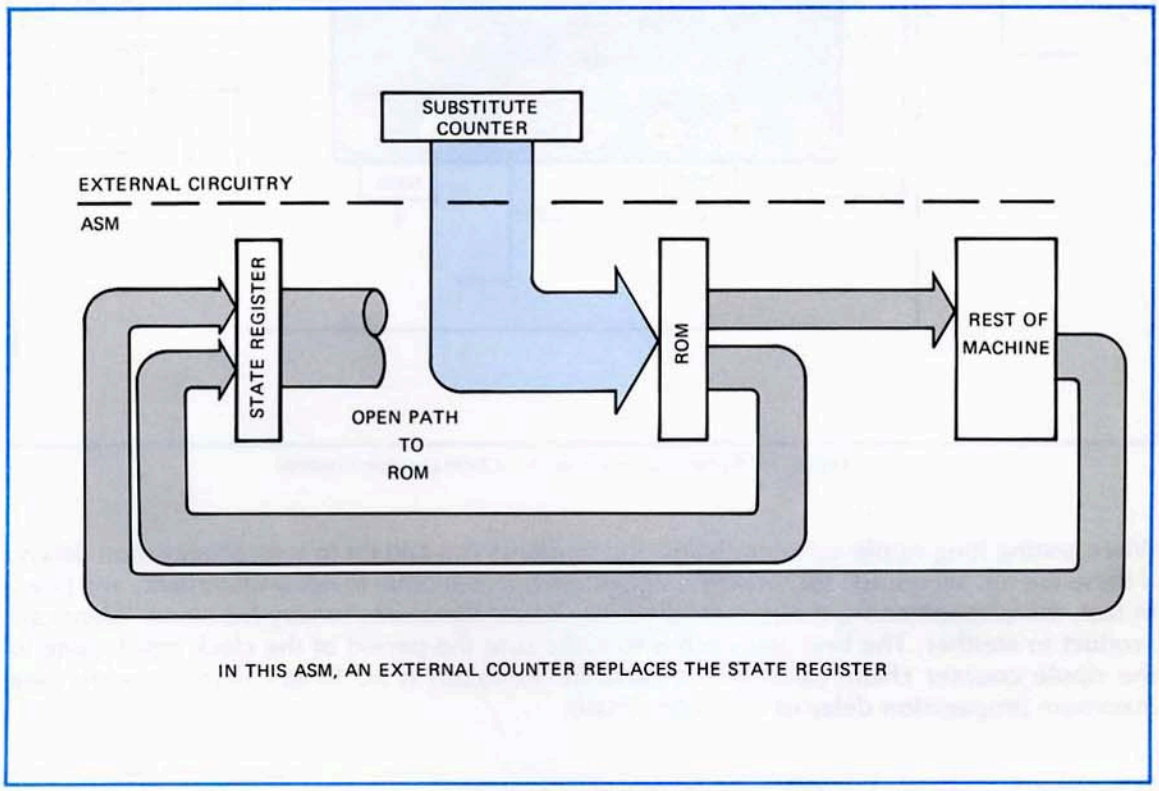


Figure 12. When the State Register cannot easily be made to free-run, an external substitute counter can force the ROM through its address states while exercising other portions of the machine.

Depending on what the ASM controls and how it does the controlling, extra ROM space may be necessary to exercise some portions of the system adequately and produce meaningful signatures. Since ASM systems generally lack the data modification capabilities of microprocessors, creating a good SA test ROM can require more words of ROM to wiggle nodes and exercise components.

If an ASM is in a microprocessor system, and the microprocessor has, or can gain control of it, the task of testing the ASM portion becomes much easier. The microprocessor, running with an SA test program can be used to manipulate and control the state pointer or counter and then exercise much or all of the ASM controlled circuitry. In a sense, the ASM becomes a programmable stimulus, with the programming data coming from the microprocessor.

Sequencers are a class of ASMs in which the A (algorithm) is absent. A sequencer simply goes from one state to the next, progressing along a predictable path, unencumbered by qualifiers and feedback in general. Aside from a possible wait or reset operation, sequencers have fairly predictable timing and are usually good candidates for SA. In simple sequencers, the SA test may be nothing more than letting it step through its states in a normal manner and taking signatures on the sequencer and the logic it controls.

RANDOM LOGIC

Products constructed entirely from random logic* generally derive the least benefit from the application of SA. Of course, the cost to incorporate SA into the design, and the resultant improvement in testability over standard techniques has to be evaluated for the particular product of concern. In general, SA makes sense when there is a lot of data processing going on, long serial data streams are present, or the system is clocked and synchronous. In circuits constructed from SSI and MSI parts, a conventional troubleshooting approach, using logic probes, pulsers, clips, and current tracers, might be just as effective for finding faults. Also, since logic and data flow are generally straightforward, the circuit is usually easier to understand by the person doing the servicing than microprocessor and ASM-based circuits.

MULTIPLE CLOCKS

In a system where more than one time base clock is used, (e.g., a frequency counter with one clock for the microprocessor and another for the measuring portion) synchronizing difficulties can often create SA testing problems.** Although it may be easy to synchronize multiple clocked circuits to each other (by waits, or interrupts), the signatures produced on many of the nodes become a function of more than a single clock time base, a condition not compatible with the one Clock input of the Signature Analyzer.

It's always best, for purposes of SA, to use a single time base clock for the entire system. If this cannot be done, it may be feasible to replace or slave all independent clocks to a single one during SA testing so that the overall system can be made synchronous. A disadvantage of doing this is that some of the circuitry will not be running at its normal operating speed, resulting in the loss of some dynamic performance criteria.

Another approach would be to isolate circuitry running on different time bases from each other and test them independently. Means for opening or disabling the signal paths between these circuit portions should be provided. Sometimes disabling appropriate clock circuits can accomplish the same function.

NONRESIDENT TEST PROGRAMS

If a product under test has communication capabilities with a programmable calculator or a computer, the SA test program can reside in them instead of the product. In operation, the calculator or computer can either control the SA exercise directly (e.g., take control of the address and data bus) or write an SA test program into the product's internal RAM, and then let it execute. The advantages of either of these techniques are: 1) Reduced overhead in the product, in terms of SA ROM space, jumpers, and switches, 2) Inexpensive SA program storage media (which encourages more thorough and service oriented testing to be done and reduces the software development effort extended toward minimizing memory size requirements), and 3) The ability to revise or upgrade the SA test program in the future without impacting product hardware or firmware. The disadvantages are that: 1) Much of the product may need to be working before the SA program can be run, 2) If the product runs the SA program from its RAM, this must be large enough to hold it, and 3) A calculator or computer is required and must be available to test the product.

---

*For purposes of this discussion, random logic refers to non bus-structured circuits of traditional combinatorial and sequential design.

**Do not confuse multiple system clock with the multiphase clocks that are used for many μps. Multiphase clocks synchronized to the same time base present no problem to the Signature Analyzer.

Section VI

Documentation

GENERAL INTENT

The general intent of a product service manual is to provide a sufficient understanding to the reader so that he can perform calibrations and diagnose and repair failures. How the manual takes form depends on the nature of the product, techinques used to adjust and troubleshoot it, and the technical level of competence assumed of the reader.

When SA is included in the troubleshooting procedure, it is important to explain the SA concept in the service manual. Since it is a new technique, the person writing the product service manual should take extra efforts to explain the principles, mechanics of use, and benefits derived from using SA to do field service repair on the product. He should explain to the service technician doing the repair how SA really does make the servicing of complex and confounding equipment much easier, and that a great deal of "shotgunning" can be eliminated. This is especially true of the technician who may have had an unpleasant experience with an inadequately documented microprocessor-based product. The initial impression and the first experience of using SA can make or break its acceptance as a troubleshooting tool by a service technician. Therefore, the importance of doing a good job of documentation the first time out cannot be over emphasized. An introduction to the SA concept, extracted from the HP 3437A Service manual, appears in Figure 13.

DOCUMENTATION TECHNIQUES

With the SA technique, the signatures provide the essential troubleshooting information, and there are many ways to document them in a service manual. The sections that follow describe some of these.

5-89. Executing nonsequential program instructions (typical event), causes changing data patterns to be present within the logic circuitry. To overcome this situation, a unique method (developed by Hewlett-Packard) is used to troubleshoot the 3437A microprocessor based logic circuits. The method is referred to as Signature Analysis (SA).

5-90. The Signature Analysis technique forces the microprocessor to continuously execute test routines. This results in continuous repetitive data patterns to be present at data nodes throughout the logic section.

5-91. A Signature Analyzer (used to probe the logic circuit data nodes) identifies the repetitive data patterns by generating a signature (4 digit alpha-numeric code) characterizing the accumulative data pattern occuring over a specific period of time.

5-92. The test signatures obtained are then compared to reference signatures (signatures generated by an instrument that is known to be in proper working order). Corresponding signatures (test & reference) imply that the section of the logic circuit that is exercised by the test routine, is functioning properly. If corresponding signatures are not obtained, then that particular section of the logic circuit is not functioning properly, and service procedures outlined in the troubleshooting flowcharts are followed.

Figure 13. Introduction to SA Extracted from HP 3437A Service Manual.

The following signatures are for the outguard Display circuits. The signatures are taken with the start/stop inputs of the Signature Analyzer connected to A3TP3.

NOTE

**To obtain this signature, a 10K resistor must be connected between the 5 volt TP and the probe tip of the signature analyzer.*

***To obtain this signature, press and hold the front panel MATH OFF key.*

†To obtain this signature, press and hold the front panel LOCAL key.

††To obtain this signature, press and hold the front panel DCV key.

To check for proper signature analyzer connections verify signature of +5V test point is 02C2.

Pin		Signature
U46	8	5AP6
	9	CUA4
	10	4F88
U49	1	66H6†
	2	6464†
	3	Probe Tip Blinks
	4	Probe Tip Blinks
	5	3AAA
	6	3818
U50	1	66H6†
	2	3818
	3	6F34†
	4	401F††
	5	6464†
	6	42AP††
	8	401F
	9	42AP
	10	42AP
	11	66H6†
	12	6464†
U51	2	42AP
	3	CF21*
	4	U65A*
	5	31CU
	6	7AAU*
	7	3AAA
	9	F541
	10	AU99
	11	OC52*
	12	1U38
	13	6UAO*
	14	H8CH*
	15	6464
U52	5	02C2
	6	0000
	8	0000
	9	02C2
	10	66H6†
	11	6464†
	12	02C2
	13	0000
U53	8	330H
	9	31CU
U54	2	5HAF
	3	6UAO*
	4	H8CH*
	5	FU84
	6	1P29*
	7	CUA4
	9	AC4A
	10	4F88
	11	U478*

Pin		Signature
U56	1	3F5A
	2	Probe Tip Blinks
	3	6144
	4	Probe Tip Blinks
	5	A874
	6	F45A
	7	PACP
	9	5381
	10	1P93
	11	74AP
	12	OPF4
	13	6757
U57	1	330H**
	3	401F††
	4	6464†
	6	6464†
	7	6464†
	9	6464†
	14	401F††
U58	3	6464†
	4	6464†
	6	6464†
	11	6464†
	13	6F34†
U59	3	UP6C
	6	2H84
	8	F400
	11	543C
U60	3	3071
	6	U478
	8	55H6
	11	9A99
U62	1	FU84
	2	CUA4
	6	4F88
	7	5HAF
	9	A29F
	10	904A
	11	6836
	12	6863
	13	U5A4
	14	5AP6
	15	P579
U63	2	6757
	3	CF21*
	4	U65A*
	5	6144
	6	7AAU*
	7	35FA
	9	AC4A
	10	54A7
	11	0C52*

Pin		Signature
U64	1	02C2
	2	0066
	3	U478*
	4	1961
	5	CH2H
	6	F62P
	7	60C6
	9	HA33
	10	U395
	13	1P29
	14	0000
	15	0207
U65	1	02C2
	2	257C
	3	CF21*
	4	U65A*
	5	38U4
	6	7AAU*
	7	2690
	9	60C6
	10	1AHF
	11	0C52*
	12	HP75
	13	6UAO*
	14	H8CH*
	15	63A2
U66	1	02C2
	2	H9C8
	3	CF21*
	4	U65A*
	5	9CC8
	6	7AAU*
	7	21C2
	9	F62P
	10	25CC
	11	OC52*
	12	142F
	13	6UAO*
	14	H8CH*
	15	A19P
U67	1	02C2
	2	U6C1
	3	CF21*
	4	U65A*
	5	AH7A
	6	7AAU*
	7	8FHP
	9	1961
	10	A2C1
	11	OC52*
	12	P6AA
	13	6UAO*
	14	H8CH*
	15	865H

Pin		Signature
U68	1	02C2
	2	895P
	3	CF21*
	4	U65A*
	5	4P3H
	6	7AAU*
	7	F9A4
	9	HA33
	10	86C7
	11	OC52*
	12	379H
	13	6UAO*
	14	H8CH*
	15	58P8
U69	1	02C2
	2	3H62
	3	CF21*
	4	U65A*
	5	AAHH
	6	7AAU*
	7	709A
	9	CH2H
	10	P2P7
	11	0C52*
	12	2851
	13	6UAO*
	14	H8CH*
	15	052A
U70	1	02C2
	2	766P
	3	CF21*
	4	U65A*
	5	26PP
	6	7AAU*
	7	086F
	9	U395
	10	F96P
	11	OC52*
	12	O50C
	13	6UAO*
	14	H8CH*
	15	2867
U71	2	A29F
	3	P579
	6	6863
	8	U5A4
	11	904A
	13	6836
	15	5AP6
	17	54A7

Figure 14. Signature Table

ANNUNCIATING SCHEMATICS

Perhaps the most novel documentation approach is similar to the "Sams Photofact®" system. Just as radio and TV schematics can be annunciated with voltages and scope waveforms, digital schematics can be annunciated with signatures. In both cases, faulty operation can be detected and traced back to the offending node. This technique can be used very effectively when the digital circuitry is fairly straightforward, understandable, and without many feedback loops, (e.g., many sequencers and random logic circuits). In more complex logic data structures, such as microprocessors and ASMs, the schematic alone does not allow much logic tracing to be done. In these cases, signatures on schematics might still be desired, but additional documentation to designate appropriate test loops, jumper connections, and Signature Analyzer hook-ups must be supplied in order to diagnose the actual fault source.

In generating schematics, it's good practice to try to use the same topology as the board for component placement and to show inputs going into the left and outputs coming out of the right side of a component. Signatures can be placed on just the signal source, (or driving IC pin) or contain an asterisk or a different color or type style to distinguish driving pins from receiving ones. When more than one valid signature may appear on the same node, (depending on a particular test mode) multicolors or typestyles can be used, or test numbers can be added to identify a particular signature.

It's easy to see that schematics can readily become cluttered with signatures. If troubleshooting of the circuit can be done directly from the signatures on the schematic, (e.g, random logic) this is a small price to pay; even if the size of the schematic must be enlarged. But, if a troubleshooting tree is required in conjunction with the signatures, it may make sense to take the signatures off the actual schematic. Another advantage of doing this is that if any signatures are likely to change because of a future product refinement, the schematic already in print will still be valid (assuming a hardware change was not also made on that particular circuit).

SIGNATURE TABLES

Lifting the signatures off the schematic and placing them in signature tables can provide a more flexible and less costly documentation approach. Figure 14 is an example of a signature table. Multiple signature tables can be created to reference various test loops which may produce a unique set of signatures on the same IC. A troubleshooting tree or flowchart could then reference specific signatures in the table thus reducing, and in some cases even eliminating, the need for the schematic in order to find a particular fault. And updating signature tables is generally less costly than any of its alternatives.

SIGNATURE MAPS

Another technique for documenting signatures is to make a signature map of the board on a component locator. This combines the finding-the-test-point and looking-up-the-signature operations into one. The disadvantage comes when high density boards result in small hard-to-read signature notations.

Taking this approach one step further, one could make cardboard or plastic templates that, when placed on a circuit board and aligned with it, provide location **and** signature information right on the board. Going even further, signal sources can be indicated and a test flow sequence can be imprinted on the template with arrows directing the technician to successive test points. See Figure 15.

®Trademark of Howard Sams Publications

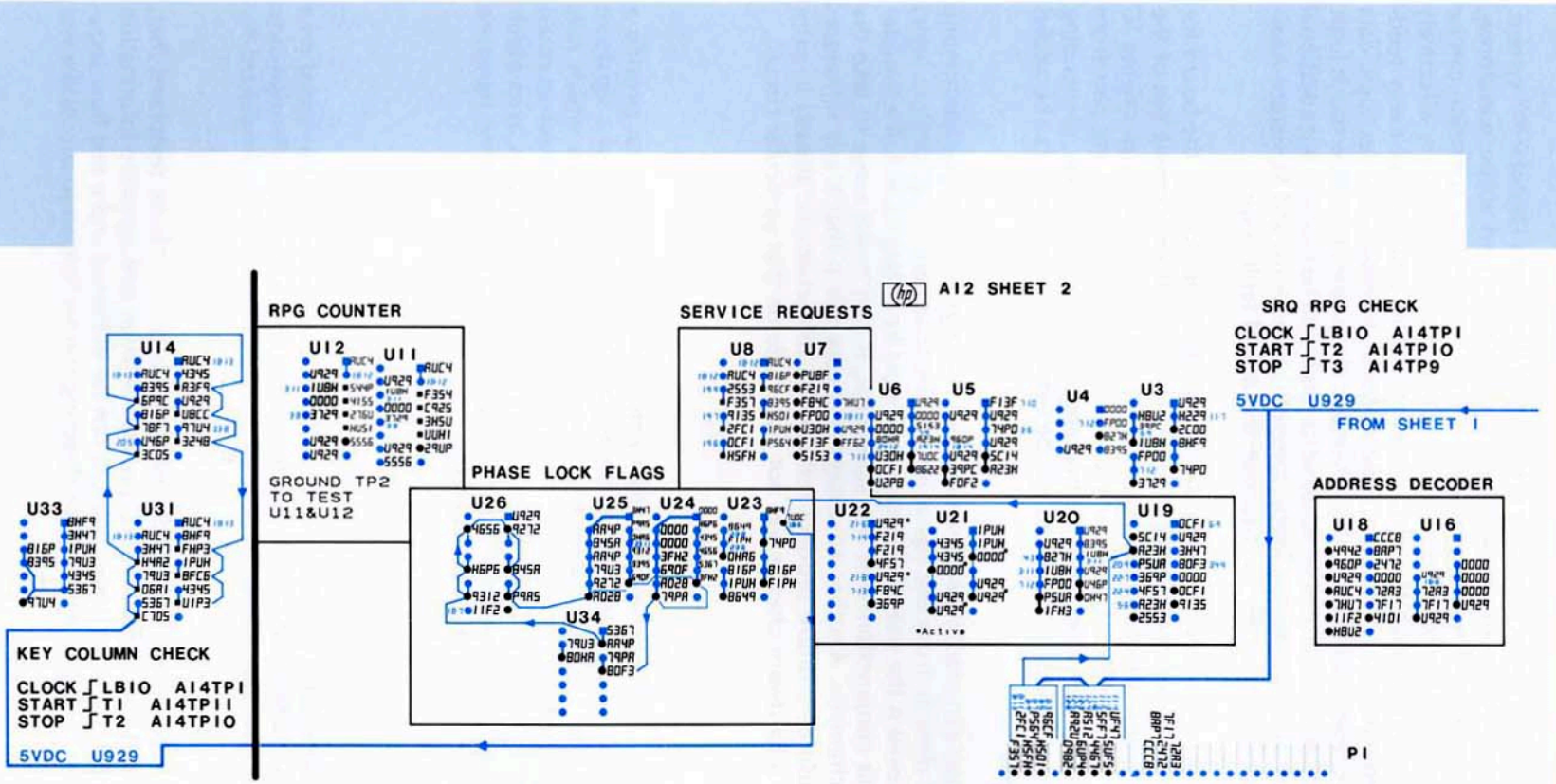


Figure 15. A TROUBLESHOOTING TEMPLATE placed on the back side of the proper P.C. board indicates test set-up instructions, IC pin signatures, a test flow "verification path", input pins ● , output pins ● , and the sources of the signatures by IC and pin number (e.g., 7–10 designates the signature source to be pin 10 of IC7).

Whatever signature documentation approach is selected, it is an advantage to minimize the number of components that determine any one signature. There are two reasons for this: 1) The job of finding the fault is made easier because fewer components need to be considered as the possible cause of a bad signature and 2) A future change in hardware or firmware will produce the minimum number of signature changes and make the job of updating the documentation less difficult.

TROUBLESHOOTING TREES AND HALF-SPLITTING

Traditional troubleshooting trees or flow diagrams provide a useful aid to service repair. Actual signatures or reference to signature tables can be included on them. Figures 16 and 17 are portions of troubleshooting diagrams used in the HP 3455A and HP 3437A Voltmeters.

A technique useful in developing troubleshooting trees is half-splitting. It can provide the shortest path to the fault. In half-splitting a test point is selected roughly midway through the circuit where there is an equally likely chance that a fault will exist before as after that point. A good measurement (signature) means that all of the circuitry up to that point is probably good (depending on how much of it gets exercised) and that the fault exists further downstream. A bad signature indicates the reverse. Once the bad half is determined, it too can be half-split, with the process being repeated until the actual fault is found.

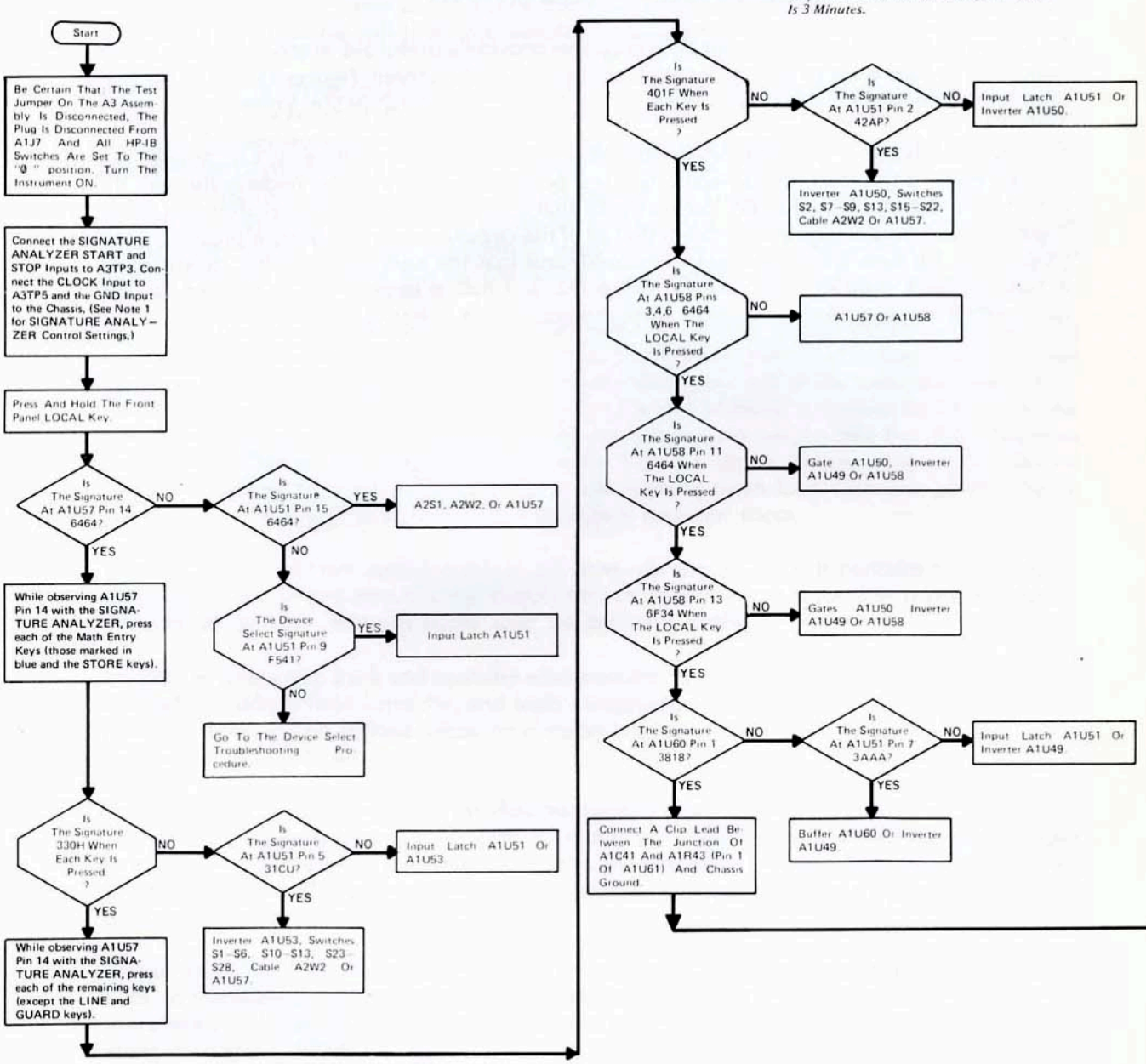
NOTE 1

The SIGNATURE ANALYZER Controls Should Be Set As Follows For This Test:

LINE . . . . . .ON (IN)
START. . . . . ⌝(IN)
STOP. . . . . ⌞(OUT)
CLOCK. . . . . ⌞(OUT)
HOLD . . . . . .OFF (OUT)
SELF TEST . . . . .OFF (OUT)

NOTE 2

A Character, Starting At The Most Significant Digit, Is Strobed Across The Display. The Characters Displayed Are 1,2,3,4,5,6,7,8,9,|,|,",Ł,L, Blank And Period. Each Character Is Strobed Across The Display Twice. The Decimal Point Accompanies The Character On The Second Strobing Sequence. Also, In The Least Significant Digit, The Decimal Point Is Lit On Each Strobe. The Only Meaningful Displaying Of The + And - Signs Is Before The Number 0 And 1 Start Their Display Sequence. The Time Required To Run The Complete Test Is 3 Minutes.



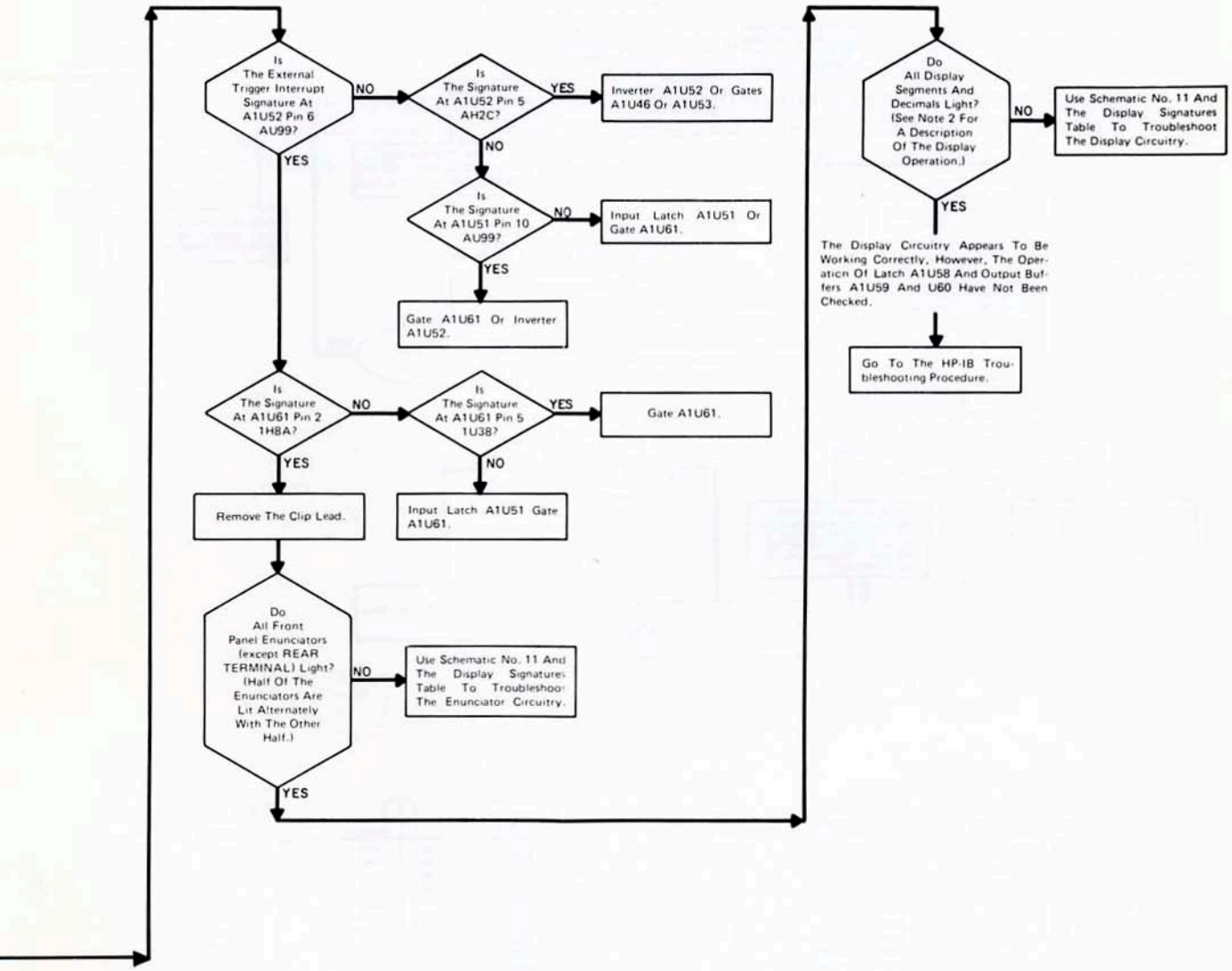
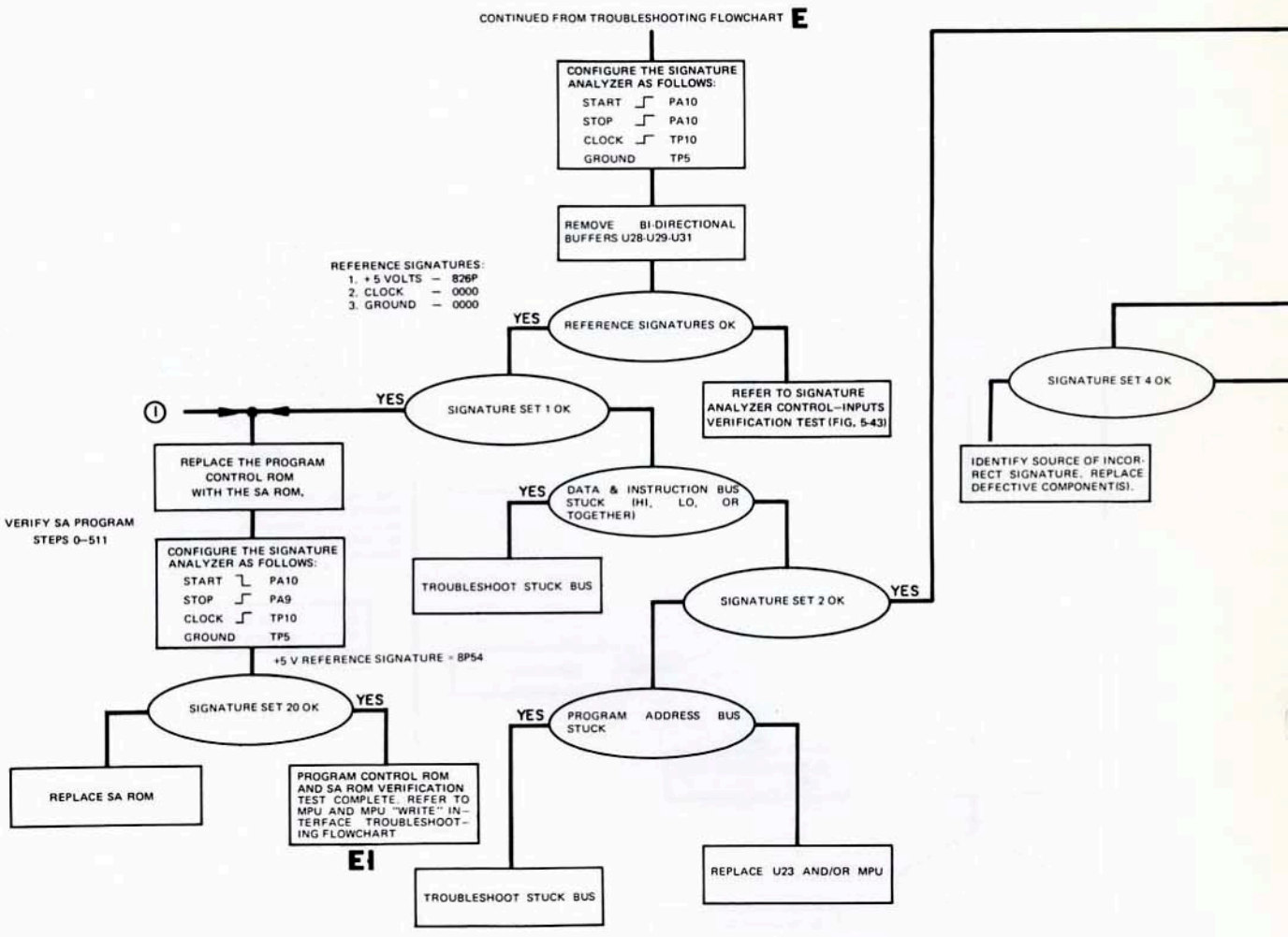


Figure 16. Signatures Included on Flowchart. Refer to HP 3455A, Figure 8-69.



(1)

Test Point	Signature
D0	7HH1
D1	9F63
D2	67FU
D3	P7F0
D4	99H5
D5	947F
D6	F487
D7	2535

(2)

Test Point	Signature
PA0	2A1F
PA1	A206
PA2	C133
PA3	8P3U
PA4	3319
PA5	7C47
PA6	C25F
PA7	5H21
PA8	19H6
PA9	HP66
PA10	U81P

(3)

U41	Signature
9	HH8A
10	AFPF
11	U75C
13	A95C
14	3F2H
15	FFU8
16	5055
17	60PH

(4)

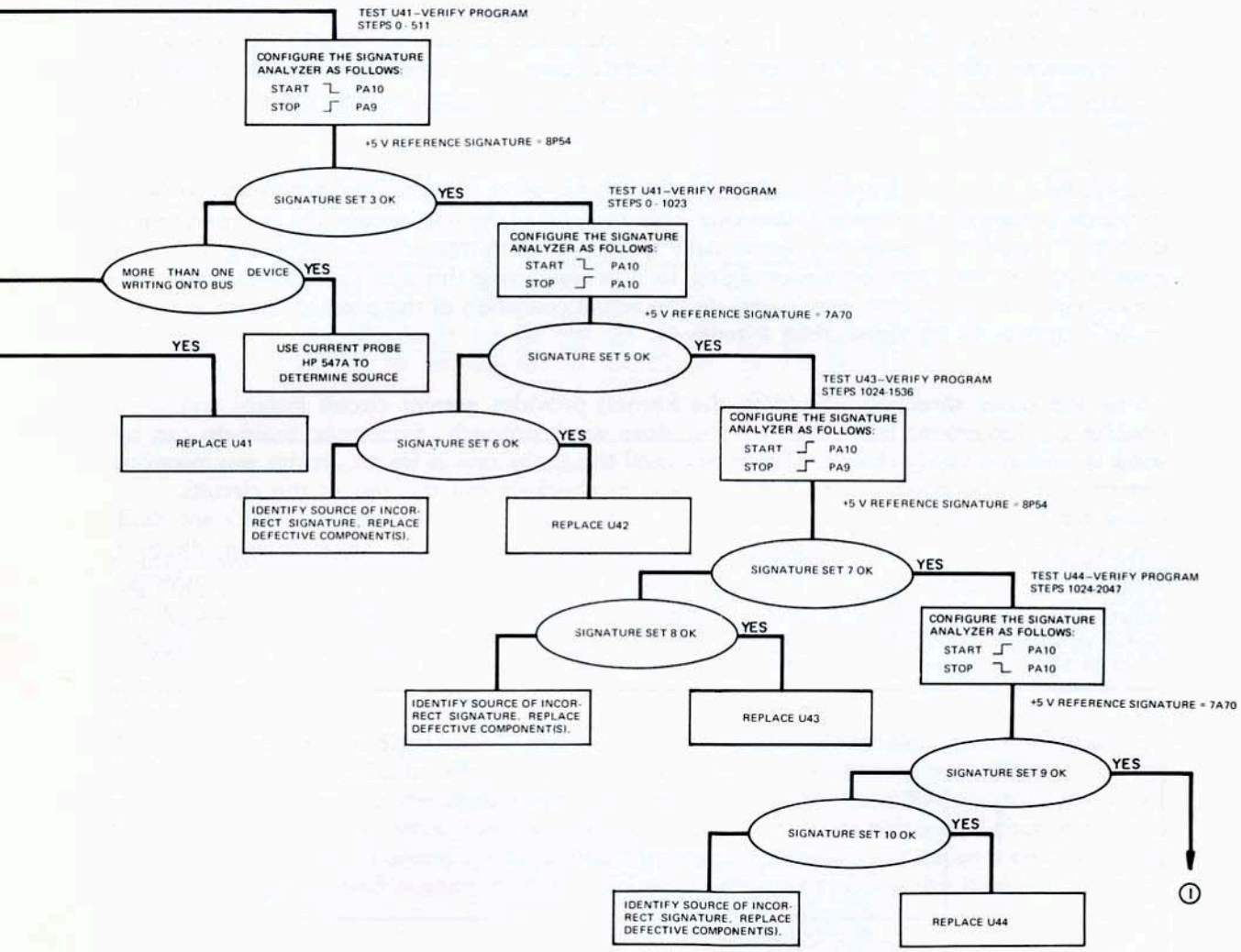
U41	Signature
1	2946
2	F61C
3	0108
4	3A9A
5	H10F
6	HH53
7	0863
8	29PP
12	0000
18	8P54
19	8P54
20	0000
21	0000
23	4596
24	8P54

(5)

U42	Signature
9	1P23
10	HP98
11	62PP
13	HHFU
14	C830
15	1A3U
16	P870
17	2U4F

(6)

U42	Signature
1	9635
2	0772
3	4U2A
4	4442
5	P030
6	H0AA
7	HA07
8	C21A
12	0000
18	7A70
19	8P54
20	0000
21	0000
23	1734
24	7A70



(7)

U43	Signature
9	C4AU
10	CU0P
11	69HA
13	FF02
14	5194
15	2CFP
16	FC26
17	23F8

(8)

U43	Signature
1	2946
2	F61C
3	0108
4	3A9A
5	H10F
6	HH53
7	0863
8	29PP
12	0000
18	8P54
19	8P54
20	0000
21	0000
23	4596
24	8P54

(9)

U44	Signature
9	1H91
10	57CP
11	1825
13	C2H7
14	17A4
15	3AUP
16	HU21
17	F955

(10)

U44	Signature
1	9635
2	0772
3	4U2A
4	4442
5	P030
6	H0AA
7	HA07
8	C21A
12	0000
18	7A70
19	8P54
20	0000
21	0000
23	1734
24	7A70

(20)

SA ROM	Signature
9	6AU4
10	A5U2
11	135A
13	5P48
14	F06P
15	9CF3
16	AA66
17	A369

Figure 17. Flowchart with Signatures Specified on Tables (Sets). Refer to HP 3437A, Figure 5-34.

The troubleshooting tree is not a perfect solution and it fits some types of hardware better than others. When there is a clear signal propagation path with minimal feedback, half-splitting makes the most sense. However, even in microprocessor systems, where the signal paths can be anything but clear, the principle of half-splitting can be employed to good advantage. Figure 18 illustrates possible first and second half-splits for a typical μp system. The first half-split may best occur between the processor (including all the bus components) and the rest of the system. In designs with both analog and digital circuits, the first half-split might be at the interface between them.

Half-splitting should be compared with working backwards, (toward the kernel) and working forwards, (enlarging the kernel). Working from the end of the line, where a faulty condition is detected, toward the beginning, (essentially from outputs to inputs) while watching for a first good measurement may be discouraging to a person using this approach. Taking one bad measurement after another may overstate the actual condition of the product under test, and make it appear to be worse than it really is.

Going the other direction, (building the kernel) provides greater circuit insight and gives positive reinforcement from circuitry that **does** work properly. Systematic build-up can be used to add and verify chunks of circuitry until the faulty one is found. In this way portions determined to be good can be used to assist in checking out the rest of the circuits.

"The fault is always found at the last place you test." When working forward or backward, there is always the possibility that the fault may be at the other end. This is why some levels of half-splitting should be used to reduce test steps. The better the troubleshooting procedure, the faster the fault can be found, and the lesser will be the temptation or need to "shotgun."

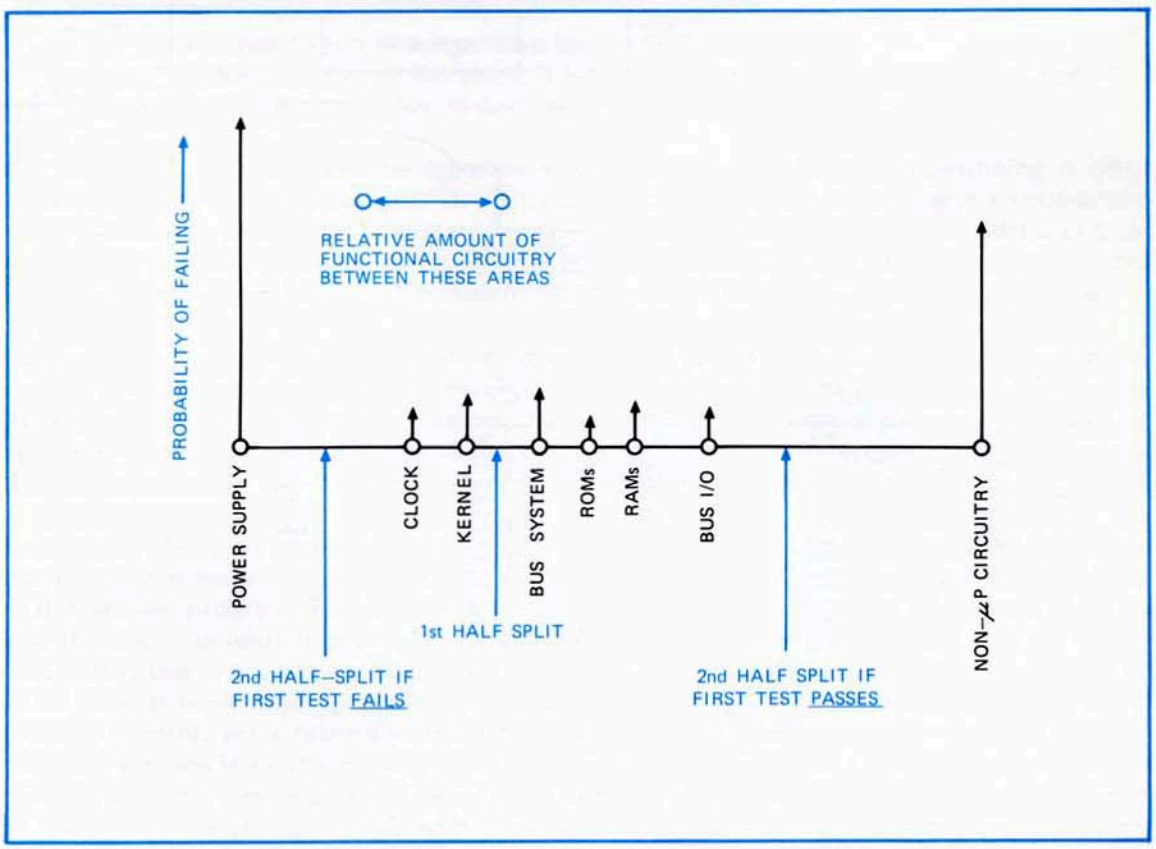


Figure 18. Half-splitting a μP System into areas of equally likely failure probabilities can produce optimum troubleshooting efficiency.

FAULT DICTIONARIES

Fault dictionaries are tables of characteristic "bad" signatures that can be useful in certain situations for detecting specific component faults. Their most practical application comes when they are used in conjunction with special SA test loops fashioned to isolate faults to specific elements, (e.g., a loop-on-fail test to identify a bad ROM as discussed in the ROM section).

It is also possible to catalog the signatures produced on a selected number of key nodes when other points in the circuit are shorted high or low. Then, if one of these points becomes stuck in the field, the signature it produces can be looked-up in the "fault dictionary" and the bad node be identified. Further, by connecting nodes to test points on the PC board that are accessible when the board is still plugged into the product, the need to use extender boards to access test points can be reduced. There are three major disadvantages to creating a fault dictionary of this type: 1) It takes extra time to gather all the signature data (and this will have to be repeated if circuit changes are made), 2) The size of the dictionary can become unwieldy, and 3) Not all types of faults can be localized this way, (e.g., multiple faults and functional failures within MSI and LSI devices). Fault dictionaries can be major aids in troubleshooting, but only if done efficiently, and with consideration for the user. The highest figure-of-merit for a fault dictionary can be achieved by one that uses the minimum number of signature entries to pinpoint the maximum percentage of possible faults.

DOCUMENTATION UPDATES

As elegant and clever as the electronic product design and documentation may be, "refinements" usually appear after the product documentation is finished. This is when extra attention to documentation technique can pay off in terms of minimizing the amount of time it takes to update the changes, and how many revision sheets get generated. It makes sense to analyze the probabilities of changes and their resulting impact on the manuals before they happen. Also, since the signatures may depend on a product series number, a board number, revision number, and even a ROM part number, care must be taken to insure accurate and reliable documentation of signatures which will add credibility to the service manual and instill user confidence.

DOCUMENTATION FOR PRODUCTION

When SA is used for production troubleshooting, the field service test and repair procedure is usually adequate. But, considering the fact that many more products fail in production than in the field, (especially multicomponent faults) it might make sense to optimize the troubleshooting procedure to the higher volume, more specialized atmosphere found in production. It may even be desirable to run more specialized test stimulus programs and use supplemental test fixtures to capitalize more fully on the advantages SA has to offer to a production test environment. Even if an automatic board tester is available, SA testing can relieve the load placed on it as well as provide in-product dynamic testing, lower test set-up time, and faster turnaround time than a heavily scheduled automatic board tester. Further, SA can help find faults on boards that elude the board tester, (e.g., timing, interface, and interconnect) but show-up once the boards are installed in the product. This can prevent the fruitless recycling of those "bad" boards that keep getting called "good" by the board tester.

Section VII

The Economic Tradeoffs

The incorporation of SA into a product affords the designer an opportunity to make an economic decision based on a myriad of tradeoff factors. The decision is not just "to use it or not to use it," but also, "to what degree." Table 1 illustrates some of the economic tradeoffs involved in adding SA to a typical microprocessor-based product (product A), and to one where a high degree of serviceability is a design requirement (product B).

Table 1. Economic Tradeoffs. In Product A, SA was included to improve serviceability. In Product B, SA was included to lower the cost of fulfilling a good serviceability requirement.

Cost Factor	Product A	Product B
Product Development	↑	↓
Service Documentation Development	↑	↓
Manufacturing Parts and Assembly	↑	↓
Factory Test and Repair	↓	↓
Warranty and Service Support	↓	*

↑ increases cost
↓ decreases cost
* Defined to be the same

Some economic factors to consider when determining the effective return on investment for adding SA and other service related capabilities include:

- Warranty cost goal
- Product price and profit
- Anticipated failure rate
- Cost of ownership
- Cost of field repair
- Acceptable down-time
- Board exchange inventory and support costs
- Service organization topography
- Volume and distribution of product
- Return on investment objectives of company
- Design time available

Section VIII

A Case Study

An example of one hypothetical situation might go something like this:

Macher Electronics manufactures microprocessor controlled gaming machines for world-wide distribution. Although their primary markets are Nevada, New Jersey, Chicago, and Monaco, major (but discrete) consumption exists in other global communities.

Macher maintains company owned service centers in most of their primary market regions and franchises service to carefully selected independent service contractors. When the company management was introduced to the Signature Analyzer and the SA concept, they were quick to realize its potential. For them it meant that costly, (and for them, sometimes risky) board exchange programs could be replaced by field repair for all their new microprocessor-based products.

Macher's design engineers give considerable attention to product reliability, since field failures can be very costly to the machine operator should false payoffs occur. The most important asset Macher has going for it is its reputation for reliability. Their customers know that this, along with a good service contract, is more important overall than the 90 day warranty agreement itself or the somewhat lower selling prices of competitive products.

Their latest product under development, "Schlameil" is typical of their present trend in microprocessor-based products. It will use an Intel 8085 with 12K words of ROM and 2K words of RAM. The microprocessor will control the six 7-segment displays, 23-indicator lamps, 5 solenoids, 3 switches, a tone generator output circuit, and a coin sensing mechanism. A self-test verification routine will be run continuously whenever the Schlameil is not in play. Only when the routine passes will the out-of-order lamp stay off, the pay-off interlock open, and the coin slot accept coins. Further, whenever the coin box is emptied, additional performance verification programs are run.

Over a 3-year expected product lifetime Macher expects to sell about 3000 Schlameils worldwide at an average selling price of $2400. Of this figure, $550 represents unit factory cost, $200 marketing, $50 warranty, and $900 general and administrative; leaving $700 profit. The estimated project cost is $250K, yielding a return factor (net profit/cost) of 8.4.

Most of the expected field failures are expected to occur in the mechanical parts and the light bulbs. Often these failures can be remedied by a squirt of oil, a simple adjustment or a lamp replacement; all capable of being performed by the purchaser. For the more serious mechanical or electrical malfunctions, service personnel must be called in.

From Macher's past experience with similar products, it was estimated that, without SA, fewer than 40% of the electrical component failures could be found by field service technicians without resorting to wholesale shotgunning. This is why, in the past, they had resorted to factory board exchange programs. Although this resulted in a high inventory commitment at both the service centers and the factory, it was considered necessary in order to keep product down-time within acceptable limits.

With SA designed into Schlameil, fewer but larger boards can be used, resulting in reduced manufacturing cost. But the big benefit is that now more than 95% of the electrical failures can be found by field service technicians, once they are acquainted with the Signature Analyzer.

There will be three options available for service repairs: 1) On-site repair, 2) Board exchange on-site with board repair at the local service facility, and 3) Back-to-the-shop repair. Macher expects options 1 and 3 to be the most common with option 2 being used only for those Saturday afternoon breakdowns, when **any** down-time can be critical. For this reason, there will still be a small number of extra boards at the service facilities; but they will be repaired there and not at the factory.

The design and service engineers at Macher estimate an additional development cost of $4K for a self-test program and another $6K hardware and software development cost to make moderately large use of SA troubleshooting capability, (much less than a good stock of service kits would have ended up costing). The service manual will end up costing more in terms of development and printing costs. But with it, a technician can make repairs down to the component level.

In terms of manufacturing cost, SA will add approximately $16 (275 words of ROM, 2 jumper packs, and 7 test pins), but the engineers argue that the savings from using fewer boards and connectors should offset this. And the "value-added" to the product by including SA will be significant. The overall net cost to repair an electronic failure is estimated to be less than 60% of what it would have been with a non-SA alternative.

For more information, call your local HP Sales Office or East (301) 948-6370 • Midwest (312) 255-9800 • South (404) 955-1500 • West (213) 877-1282. Or, write: Hewlett-Packard, 1501 Page Mill Road, Palo Alto, California 94304. In Europe, Post Office Box, CH-1217 Meyrin 2, Geneva, Switzerland. In Japan, Yokogawa-Hewlett-Packard, 1-59-1, Yoyogi, Shibuya-Ku, Tokyo, 151.

02-5952-7465 APRIL 1977 PRINTED IN U.S.A.