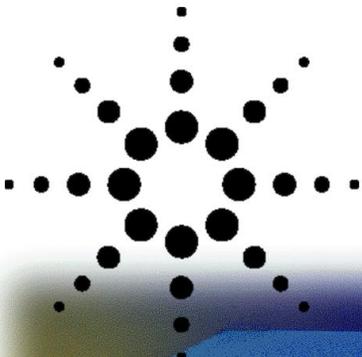


# Feeling Comfortable with Logic Analyzers

Application Note 1337



**Agilent Technologies**  
Innovating the HP Way

---

## Feeling Comfortable with Logic Analyzers

You need to simultaneously look at the inputs and outputs of a 16-bit counter to determine a timing error and you have only a 2-channel scope. How do you look at them all? You've just developed timing diagrams for a board full of digital circuitry. How do you verify them? An intermittent glitch appears to occur on one of the data lines of your microprocessor system, causing the processor to get incorrect data. You can't trigger on just the glitch with your oscilloscope. What do you use to capture and analyze it?

With the wrong tool, solving these kinds of problems can be very time consuming. Success may depend on knowing which tool can get the job done quickly. For the above problems, the best solution is the logic analyzer.

This application note is intended as a quick overview of logic analyzer basics. With it, you can cut down on the time investment needed to learn a new instrument. It doesn't cover many detailed measurements, but it does give you a good idea of what a logic analyzer can do. We talk about questions like "Why should I use a logic analyzer?" and "What will one do for me?"

## Contents

1 Oscilloscope or Logic Analyzer? . . . . .	3
2 What's a Logic Analyzer? . . . . .	5
What's a Timing Analyzer? . . . . .	6
What's a State Analyzer? . . . . .	13
3 Put Them All In One Instrument! . . . . .	18
4 How Do I Connect To My Target System? . . . . .	21
Summary . . . . .	23

## Oscilloscope or Logic Analyzer?

1

When given the choice between using an oscilloscope or a logic analyzer, many people will

choose an oscilloscope. Why? Because a scope is more familiar to most users.

However, a scope has some shortcomings that limit its usefulness in some applications. A logic analyzer may yield more useful information in many of these applications. Because of some overlapping capabilities between a scope and a logic analyzer, either may be used in some cases. How do you determine which is better for your application? The next few paragraphs give some basic guidelines.

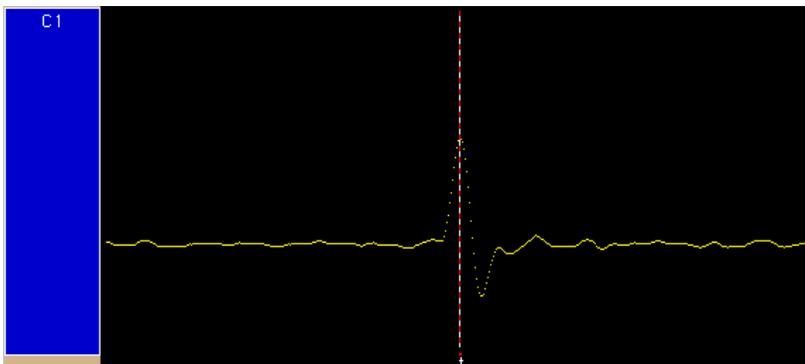
### When Should I Use A Scope?

- When you need to see small voltage excursions on your signals.
- When you need high time-interval accuracy.

Generally, an oscilloscope is the instrument to use when you need high vertical or voltage resolution. To say it another way, if you need to see every voltage excursion, like those shown in figure 1.1, you should use a scope.

Many scopes, including the new generation digitizing ones, can also provide very high time-interval resolution. That is, they can measure the time interval between two events with very high accuracy. Overall, an oscilloscope is to be used when you need parametric information.

Figure 1.1 Oscilloscope waveform



---

## When Should I Use A Logic Analyzer?

- When you need to see many signals at once.
- When you need to look at signals in your system the same way your hardware does.
- When you need to trigger on a pattern of highs and lows on several lines and see the result.

Logic analyzers grew out of oscilloscopes. They present data in the same general way that a scope does; the horizontal axis is time, the vertical axis is voltage amplitude. But, rather than providing high voltage resolution or time interval accuracy like a scope, a logic analyzer can capture and display hundreds of signals at once, something that a scope can not do.

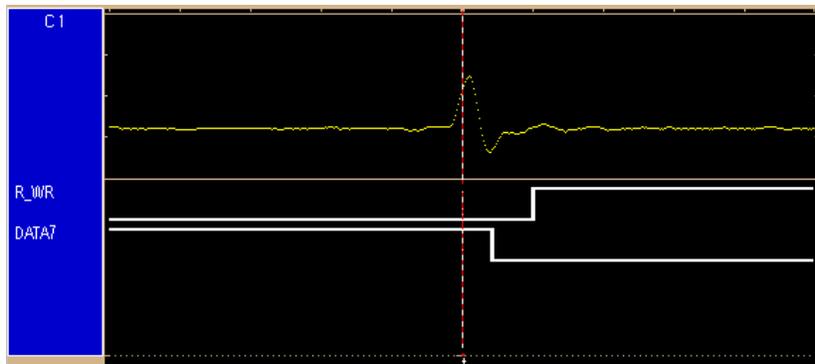
A logic analyzer reacts the same way as your logic circuit does when a single threshold is crossed by a signal in your system. It will recognize the signal to be either low or high. It can also trigger on patterns of highs and lows on these signals.

So when do you use a logic analyzer? When you need to look at more lines than your oscilloscope can show you, provided you do not need precise time interval information. If you need to look at parametric information, a logic analyzer is not a good choice (figure 1.2).

Logic analyzers are particularly useful when looking at time relationships or data on a bus – e.g. a microprocessor address, data, or control bus. It can decode the information on microprocessor buses and present it in a meaningful form.

Generally, when you are past the parametric stage of design, and are interested in timing relationships among many signals and need to trigger on patterns of logic highs and lows, the logic analyzer is the right tool.

Figure 1.2 Oscilloscope and timing waveforms



## What's a Logic Analyzer?

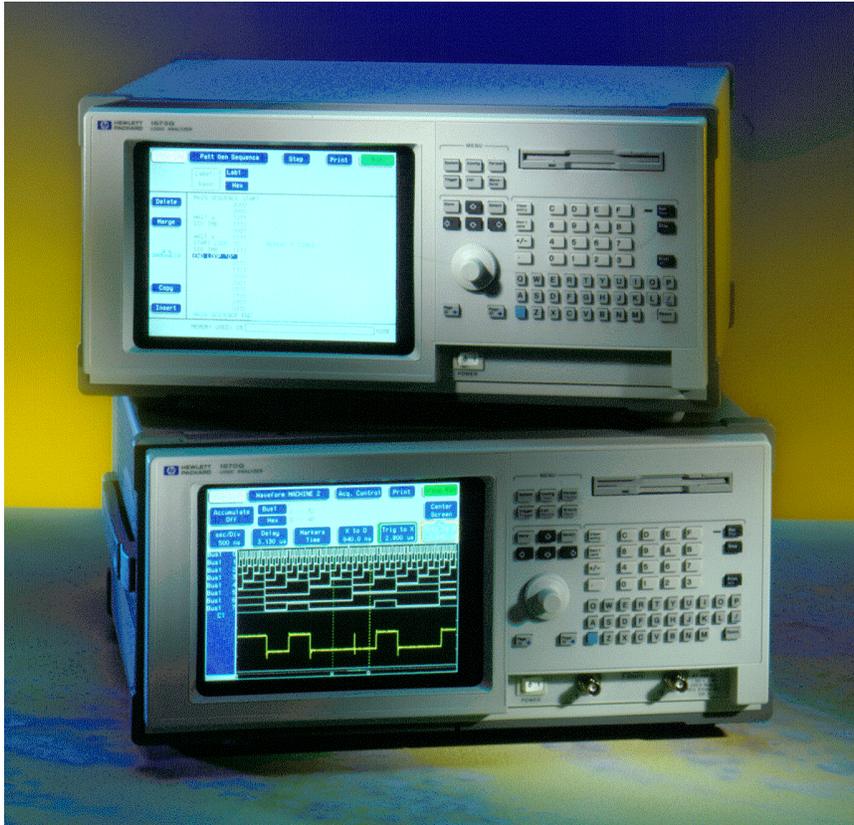
2

Now that we've talked a little about when to use a logic analyzer, let's look in a bit more detail at

what a logic analyzer is. Up to now, we've used the term "logic analyzer" rather loosely. In fact, most logic analyzers are really two analyzers in one.

The first part is a timing analyzer, while the second part is a state analyzer. Each has specific functions that we will talk about in the following sections.

Figure 2.1 An analyzer with a built in pattern generator and an analyzer with a built in scope.



## What's a Timing Analyzer?

A timing analyzer is the part of a logic analyzer that is analogous to an oscilloscope. As a matter of fact, they can be thought of as close cousins.

The timing analyzer displays information in the same general form as a scope, with the horizontal axis representing time and the vertical axis as voltage amplitude. Because the waveforms on both instruments are time-dependent, the displays are said to be in the "time domain".

### Choosing the Right Sampling Method

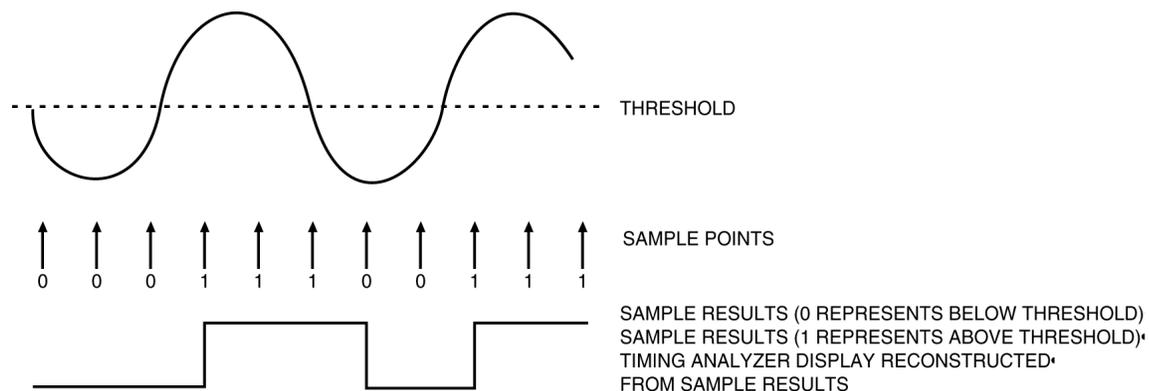
A timing analyzer works by sampling the input waveforms to determine whether they are high or low. It cares about only one user-defined voltage-threshold. If the signal is above threshold when it samples, it will be displayed as a 1 or high by the analyzer.

By the same criterion, any signal sampled that is below threshold is displayed as a 0 or low. From these sample points, a list of ones and zeros is generated that represents a one-bit picture of the input waveform.

As far as the analyzer is concerned, the waveform is either high or low – no intermediate steps. This list is stored in memory and is also used to reconstruct a one-bit picture of the input waveform, as shown in figure 2.2.

A timing analyzer is like a digitizing scope with one bit of vertical resolution. With one bit of resolution, you can display only two states – high or low.

Figure 2.2 Timing analyzer sample points



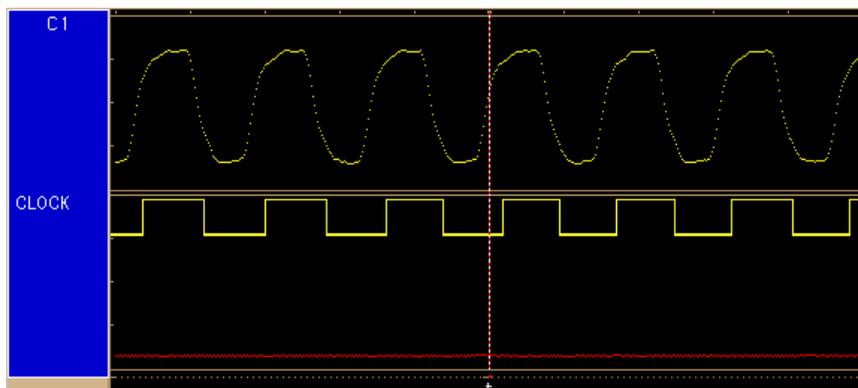
---

Notice the display shown in figure 2.3. These displays are actually the same signal (a sine wave) displayed by a digitizing scope and a timing analyzer.

This tendency to square everything up would seem to limit the usefulness of a timing analyzer. We should remember, however, that it is not intended as a parametric instrument. If you want to check rise time of a signal with an analyzer, you should use a scope. But if you need to verify timing relationships among several or hundreds of lines by seeing them all together, a timing analyzer is the right choice.

For example, imagine that we have dynamic RAM in a system that must be refreshed every 2 ms. To ensure that everything in memory is refreshed within that 2 ms, a counter is used to count up sequentially through all rows of the RAMs and refresh each. If we want to make certain that the counter does indeed count up through all rows before starting over, a timing analyzer can be set to trigger when the counter starts and display all of the counts. Parametrics are not of great concern here – we merely want to check that the counter counts from 1 to N and then starts over.

Figure 2.3 The same signal displayed by an oscilloscope and a timing analyzer



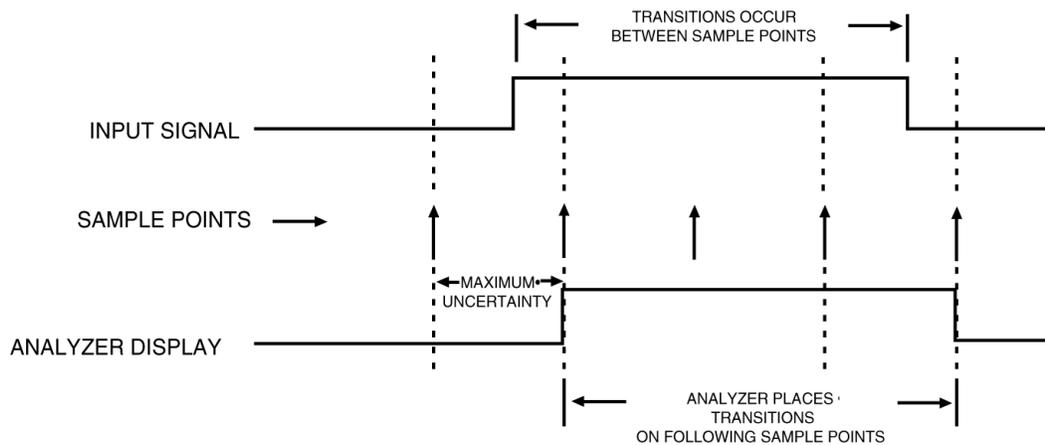
---

When the timing analyzer samples an input line, it is either high or low. If the line is at one state (high or low) on one sample and the opposite state on the next sample, the analyzer "knows" that the input signal transitioned at sometime in between the two samples. It doesn't know when, so it places the transition point at the next sample, as shown in figure 2.4. This presents some ambiguity as to when the transition actually occurred and when it is displayed by the analyzer.

Worst case for this ambiguity is one sample period, assuming that the transition occurred immediately after the previous sample point.

With this technique, however, there is a trade-off between resolution and total acquisition time. Remember that every sampling point uses one memory location. Thus, the higher the resolution (faster sampling rate), the shorter the acquisition window.

Figure 2.4 Timing analyzer sampling an input line



## Transitional Sampling

When we capture data on an input line with data bursts, like in figure 2.5, we have to adjust the sampling rate to high resolution (e.g. 4 ns) to capture the fast pulses at the beginning. This means that a timing analyzer with 4K (4096 samples) memory would stop acquiring data after 16.4  $\mu$ s and the second data burst could not be captured.

Note that we sample and store data for a long time where there is no activity. This uses up logic analyzer memory without providing additional information.

What we need to know is when these transitions occur and if they are positive or negative. Transitional timing, however, uses memory efficiently.

To accomplish this, we could use a "transition detector" at the input of the timing analyzer together with a counter. The timing analyzer will now store only those samples that were preceded by a transition, together with the elapsed time from the last transition. Thus we use only two memory locations per transition and no memory at all if there is no activity at the input. This technique is called "transitional timing" and is used in the 1660/70 family of logic analyzers.

In our example, we can not only capture the second burst, but also the third, fourth, and fifth, depending on how many pulses per burst are present. At the same time, we can keep the timing resolution as high as 4 ns (figure 2.6).

We can now talk about "effective memory depth," which equals the total time data is captured divided by the sampling period (4 ns).

**Note:** This is a conceptual description of the transitional timing technique.

Figure 2.5 Sampling at high resolution

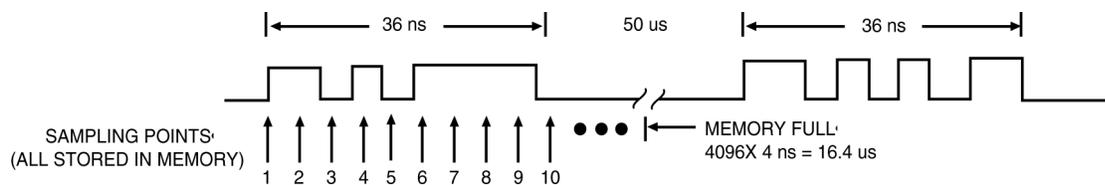
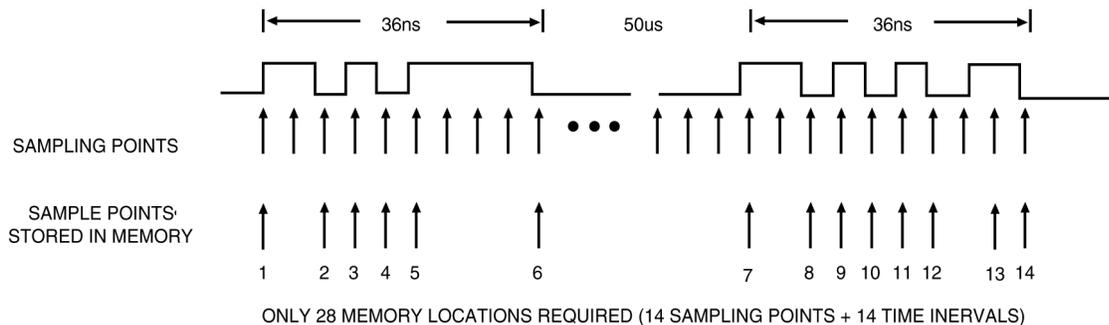


Figure 2.6 Sampling with a transition detector



---

## Glitch Capture

One headache of digital systems is the infamous "glitch". Glitches have a nasty habit of showing up at the most inopportune times with the most disastrous results. How do you capture a glitch that occurs once every 36 hours and sends your system into the weeds? Once again the timing analyzer comes to the rescue! Agilent Technologies analyzers have glitch capture and trigger capability that make it easy to track down elusive glitch problems.

A glitch can be caused by capacitive coupling between traces, power supply ripples, high instantaneous current demands by several devices, or any number of other events. Because they are difficult for most oscilloscopes to differentiate from valid transitions, a scope is generally not helpful in tracking down a glitch.

However, since a timing analyzer samples the incoming data and can keep track of any transitions that occur between samples, it can readily recognize a glitch. In the case of an analyzer, a glitch is defined as any transition that crosses logic threshold more than once between samples (figure 2.7).

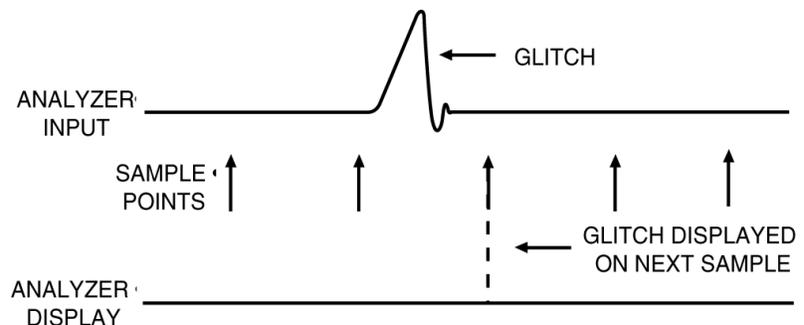
The analyzer already keeps track of all single transitions that occur between samples, as we discussed before. To recognize a glitch, we "teach" the analyzer to keep track of all multiple transitions and display them as glitches.

While displaying glitches is a useful capability, it can also be helpful to have the ability to trigger on a glitch and display data that occurred before it. This can help us to determine what caused the glitch. This capability also enables the analyzer to capture data only when we want it – when the glitch occurred.

Think about the example we mentioned in the beginning paragraph of this section. We have a system that crashes periodically because a glitch appears on one of the lines. Since it occurs infrequently, to store data all the time (assuming we had enough storage capability) would result in an incredible amount of information to sort through. Another alternative is to use an analyzer without glitch trigger capability and sit in front of the machine pressing the RUN button and waiting until you see the glitch.

Unfortunately, neither of the above are practical alternatives. If we can tell the analyzer to trigger on a glitch, it can stop when it finds one, capturing all the data that happened before it. We let the analyzer be the baby-sitter and when the system crashes, we have a record of what led up to the error.

Figure 2.7 A glitch



## Triggering the Timing Analyzer

Another term that should be familiar to oscilloscope users is "triggering". It is also used in logic analyzers, but is often called "trace point." Unlike an oscilloscope which starts the trace right after the trigger, a logic analyzer continuously captures data and stops the acquisition after the trace point is found to display the data. Thus a logic analyzer can show information prior to the trace point, which is known as negative time, as well as information after the trace point.

## Pattern Trigger

Setting trace specifications on a timing analyzer is a bit different than setting trigger level and slope on an oscilloscope. Many analyzers trigger on a pattern of highs and lows across input lines.

Notice the menu in figure 2.8. We have told the analyzer to start capturing data when channels 0, 2, 4 and 6 of "INT4" are high (logical 1) and when channels 1, 3, 5, and 7 are low (logical 0).

Figure 2.9 shows the resulting display with the line in the middle indicating the trace point. At the trace point channels 0, 2, 4 and 6 are all high while channels 1, 3, 5, and 7 are low.

To make things easier for some users, the trigger point on most analyzers can be set not only in binary (1's and 0's) but in hex, octal, ASCII, or decimal. For instance, to set the previous example in hex, the trigger specification would be 55 instead of 0101 0101. Using hex for the trigger point is particularly helpful when looking at buses that are 4, 8, 16, 24, or 32 bits wide. Imagine how cumbersome it would be to set a specification for a 24-bit bus in binary!

Figure 2.8 INT4 set to trigger on a pattern of highs and lows

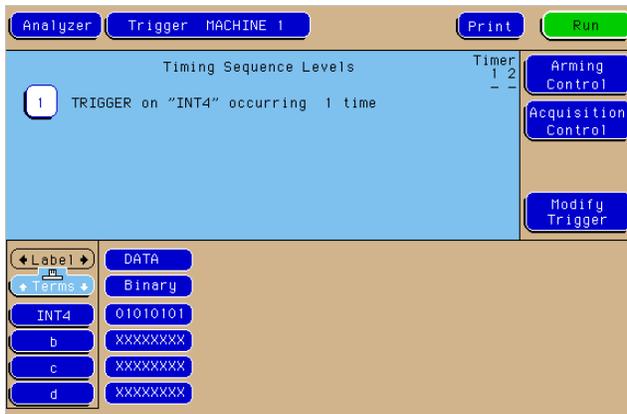
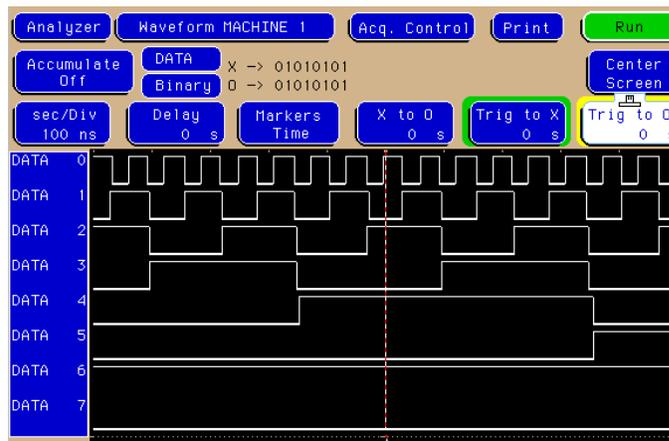


Figure 2.9 Waveform with the trace point



---

## Edge Trigger

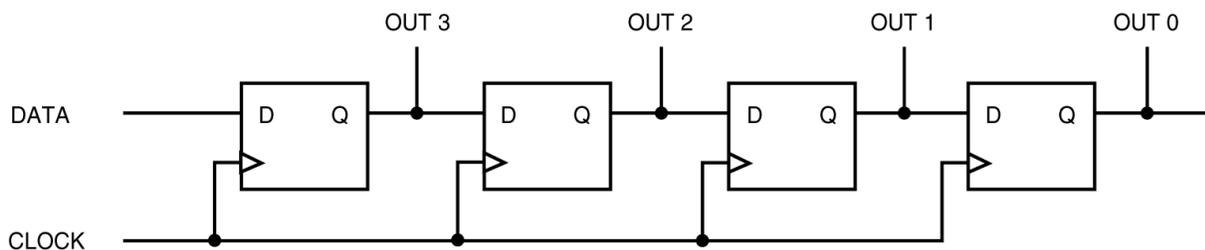
Edge triggering is a familiar concept to those accustomed to using an oscilloscope. When adjusting the "trigger level" knob on a scope, you could think of it as setting the level of a voltage comparator that tells the scope to trigger when the input voltage crosses that level. A timing analyzer works essentially the same on edge triggering except that the trigger level is preset to logic threshold.

Why include edge triggering in a timing analyzer? While many logic devices are level-dependent, clock and control signals of these devices are often edge-sensitive. Edge triggering allows you to start capturing data as the device is clocked.

As a simple example, take the case of an edge-triggered shift register that is not shifting data correctly. Is the problem with the data or the clock edge? In order to check the device, we need to verify the data when it is clocked – on the clock edge (figure 2.10).

The analyzer can be told to capture data when the clock edge occurs (rising or falling) and catch all of the outputs of the shift register. Of course, in this case we would have to delay the trace point to take care of the propagation delay through the shift register.

Figure 2.10 Edge-triggered shift register



## What's a State Analyzer?

If you've never used a state analyzer, you may think it's an incredibly complex instrument that would take a large time investment to master.

"Besides," you say to yourself, "what use could I have for a state analyzer? I design hardware."

The truth is, many hardware designers find a state analyzer to be a very valuable tool, especially when tracking down bugs in software or hardware. It can eliminate "finger-pointing" between hardware and software teams when a problem does come up. Plus the state analyzer is not any more difficult to understand than the timing analyzer.

In the first part of this application note we talked about one of two major parts of a logic analyzer – the timing analyzer. Next we will talk about the other major part of the logic analyzer – the state analyzer.

### When Should I Use a State Analyzer?

If we want to understand when to use a state analyzer, we need to know first what is a "state." A "state" for a logic circuit is a sample of a bus or line when its data is valid.

For example, take a simple "D" flip-flop, like the one shown in figure 2.11. Data at the "D" input will not be valid until a positive-going clock edge comes along. Thus, a state for the flip-flop is when the positive clock edge occurs.

Now imagine that we have eight of these flip-flops in parallel. All eight are connected to the same clock signal (figure 2.12).

When a positive transition occurs on the clock line, all eight will capture data at their "D" inputs. Again, a state occurs each time there is a positive transition on the clock line. These eight lines are analogous to a microprocessor bus.

If we connected a state analyzer to these eight lines and told it that a positive transition on the clock line is when we want to collect data, the analyzer would do just that. Any activity on the inputs will not be captured by the state analyzer unless the clock is going high.

Figure 2.11 D flip-flop

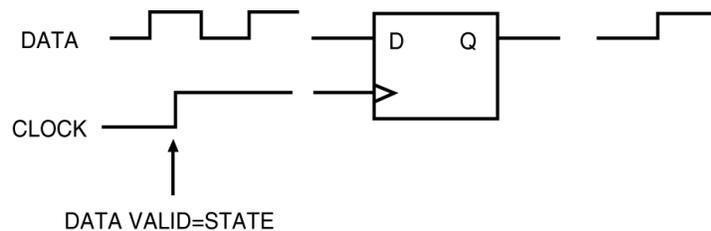
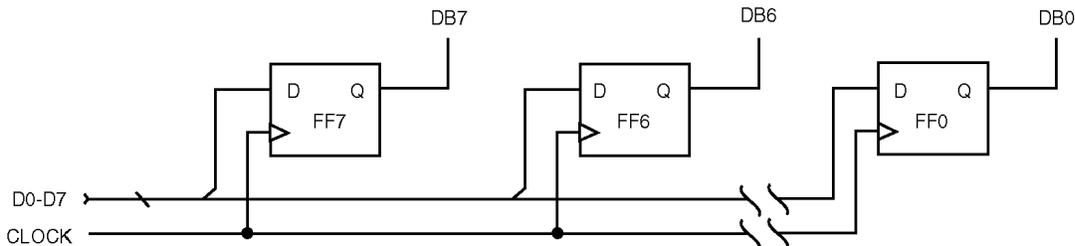


Figure 2.12 Eight D flip-flops in parallel connected to the same clock signal



This points up the major difference between a timing and a state analyzer. The timing analyzer has an internal clock to control sampling, so it asynchronously samples the system under test. A state analyzer synchronously samples the system since it gets its sampling clock from the system.

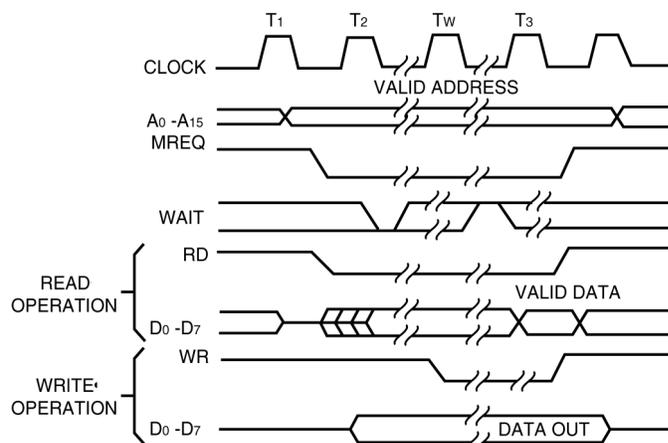
As a rule of thumb, you might remember to use a state analyzer to check "what" happened on a bus and a timing analyzer to see "when" it happened. Therefore, a state analyzer generally displays data in a listing format and a timing analyzer displays data as a waveform diagram. We have to be extremely careful not to misinterpret the data when the logic analyzer is capable of displaying state data as a waveform diagram and timing data as a listing.

## Understanding Clocks

In the timing analyzer, sampling is under direction of a single internal clock. That makes things very simple. However, in the world of microprocessors, a system may have several "clocks." Let's look at a brief example.

Suppose for a moment that we want to trigger on a specific address in RAM and see what data is stored there. Further, we'll assume that the system uses a Zilog Z80.

In order to capture addresses from the Z80 with our state analyzer, we will want to capture when MREQ line goes low. But to capture data, we will want the analyzer to sample when the WR line goes low (write cycle) or when RD goes low (read cycle). Some microprocessors multiplex data and address on the same lines. The analyzer must be able to clock in information from the same lines but with different clocks.



During a read or write cycle, the Z80 first puts an address on the address bus. Next it asserts MREQ, showing that the address is valid for a memory read or write. Last, the RD or WR line is asserted, depending on whether we are doing a read or write. The WR line is asserted only after the data on the bus is valid.

This, in essence, acts as a demultiplexer to capture an address at the proper time and then catch data that occurs on the same lines.

## Triggering the State Analyzer

Similar to a timing analyzer, a state analyzer has the capability to qualify the data we want to store. If we are looking for a specific pattern of highs and lows on the address bus, we can tell the analyzer to start storing when it finds that pattern and to continue storing until the analyzer's memory is full.

In the following example, we have set the trigger point as 406F6 (hexadecimal) (figure 2.13). In this case we want to find out what is in location 406F6, so we set the data trigger as don't cares (XXXX). This tells the analyzer to trigger on address 406F6 regardless of what the data is at that point.

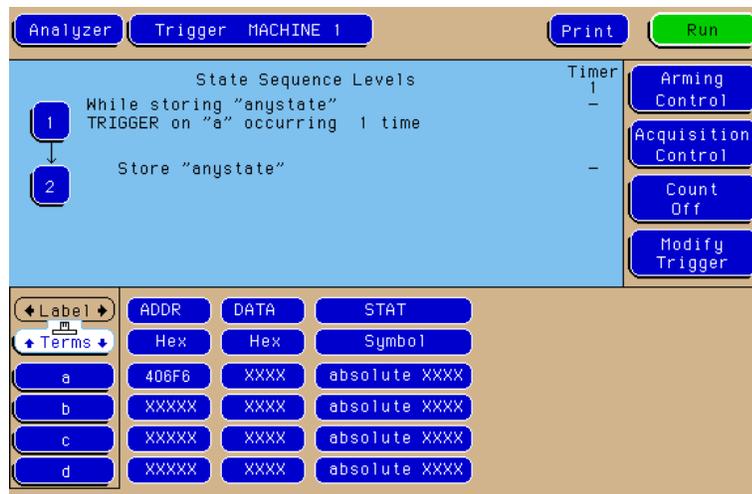
The analyzer captured address 406F6 and all following states. Notice that data is 101F at address 406F6 (figure 2.14), and that all of the information is displayed in hexadecimal format. We could display it in binary, if that is helpful. However, it may be more helpful to have the hex decoded into assembly code.

If you specify that all information on the buses is to be displayed in hex, you will get a display that resembles the one in figure 2.14.

Figure 2.14 Data captured by the state analyzer

Label>	ADDR	DATA	STAT	
Base>	Hex	Hex	Symbol	
-14	406F6	101F	absolute	DFFB
-13	406F8	4A00	absolute	DFFB
-12	0FF7E	20AC	absolute	DFF6

Figure 2.13 Trigger setup for the state analyzer



What do these hex codes mean? In the case of a processor, specific hex characters comprise an instruction. If you are very familiar with the hex codes, you may be able to look at a hex listing like the one in figure 2.14 and know what instruction is represented by it. Most of us, however, can't do that. For that reason, most analyzer makers have designed software packages called disassemblers or inverse assemblers. The job of these packages is to translate the hex codes into assembly code to make them easier to read.

For example, figure 2.14 has 101F and 4A00 shown. If we look those codes up in the Motorola CPU32 manual we find that they represent MOVE.B (move byte) and TST.B (test byte) instructions. Rather than having to look each code up, the inverse assembler does it for us. Look at figure 2.15 and notice the difference.

## Understanding Sequence Levels

State analyzers have "sequence levels" that aid triggering and storage. Sequence levels allow you to qualify data storage more accurately than a single trigger point. This means that you can more accurately window in on the data without storing information you don't need. Sequence levels usually look something like this:

```
1 find xxxx
   else on xxxx go to level x
2 then find xxxx
   else on xxxx go to level x
3 trigger on xxxx
```

Sequence levels are especially useful for getting into a subroutine from a specific point in the program.

Figure 2.15 Hex codes translated into assembly code

Label>	ADDR	CPU32 DATA	STAT
Base>	Hex	mneumonics / hex	S mbol
-14	406F6	MOVE.B (A7)+,D0	absolute DFFB
-13	406F8	TST.B D0	absolute DFFB
-12	0FF7E	data read 20xx	absolute DDF6

---

## Selective Storage Saves Memory and Time!

Sequence levels make possible what we call selective storage. Selective storage simply means storing only a portion out of a larger whole.

For instance, suppose we have an assembly routine that calculates the square of a given number. If the routine is not calculating the square correctly, we can tell the state analyzer to capture that routine. We do this by first telling the analyzer to find the start of the routine. When it does find the start address, we then tell it to look for the ending address while storing everything in-between. When the end of the routine is found, we tell the analyzer to stop storing (store no states). Figure 2.16 shows how selective storage works.

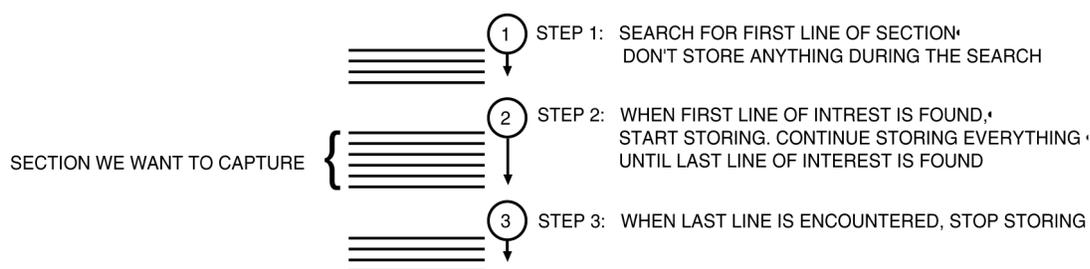
## Using Trigger Functions

Rather than defining each sequence level from scratch, you can use pre-defined trigger functions. A library of common trigger functions, such as “Find Nth occurrence of an edge” and “Find event “n” times”, provide a simple way to set up the analyzer to trigger on common events and conditions. Functions are available for both the state and timing acquisition modes.

You can also use pre-defined trigger functions as a starting point for creating custom functions. When you break down a function, you gain access to all the resource assignment fields and branching options. You can change these fields to change the trigger structure.

You might need to do this to create a custom trigger specification or to create loops and jumps in your trigger sequence.

Figure 2.16 Selective storage



## Put Them All In One Instrument!

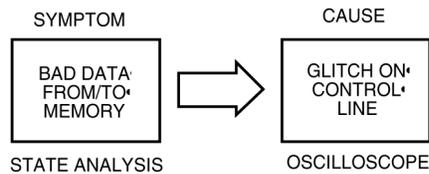
# 3

So far we have talked about oscilloscopes, state and timing analyzers and their applications. If you are designing or servicing digital hardware, you probably have applications for each one of the tools in your area. In this section we'll talk about how to use these tools together to isolate the faults in your system faster and more efficiently.

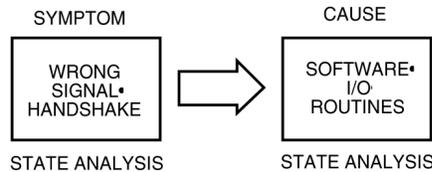
### Symptoms and Their Cause

If you troubleshoot digital circuitry you often have to ask yourself, "What causes this symptom?" It might be quite easy to identify the symptom of a fault, but you need to find the cause to fix the problem. Many times, causes and symptoms are in different domains. For example, a glitch on a memory control line can cause wrong data to be read from or written to memory. The symptom (wrong data) can be found in the data domain by using a state analyzer and triggering on the suspect memory address. The cause, however, cannot be identified in the data domain.

A glitch is only identifiable in the time domain by using either a timing analyzer or a scope. In our example, we are more interested in the size of the glitch (parameter) and therefore, the scope is the best tool for the measurement.



It is also possible that the symptom is in the time domain (e.g. a bad handshake signal on I/O lines), and the cause is in the data domain (wrong software I/O routine).



## Intermodule Measurements

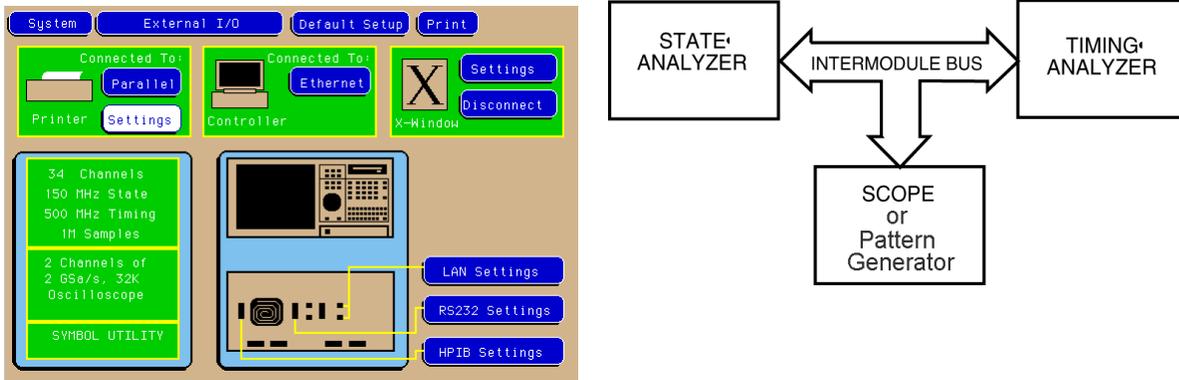
A measurement which involves more than one measurement instrument is called an "intermodule measurement."

An intermodule measurement requires that all measurement tools are integrated in a single instrument and are able to capture data simultaneously. The figure 3.1 shows the system configuration menu from a 1670G logic analyzer system with a built in oscilloscope. For the 1670G logic analyzer with a built in pattern generator, you would see the pattern generator in place of the oscilloscope.

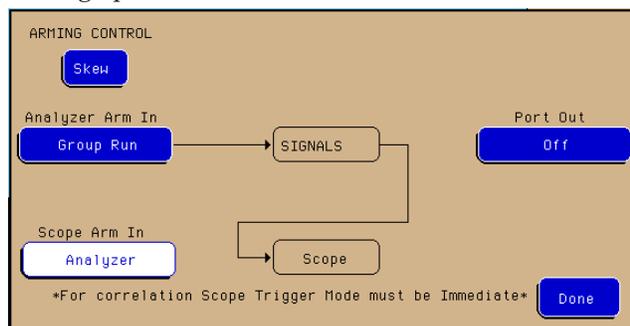
## Cross-Domain Triggering

In our examples we talked about triggering a module (state, timing analyzer or scope) on the symptom of the problem. Once the symptom occurs and the appropriate analyzer triggers, the module that monitors the cause has to start capturing data. This is achieved by arming one module from the trigger of the other module. For full functionality it is necessary that each module can receive and send trigger signals. The bus, on which these trigger signals are transmitted, is called the "intermodule bus" or IMB (figure 3.1).

Figure 3.1 System configuration menu and the intermodule bus



## Setting up an intermodule measurement



---

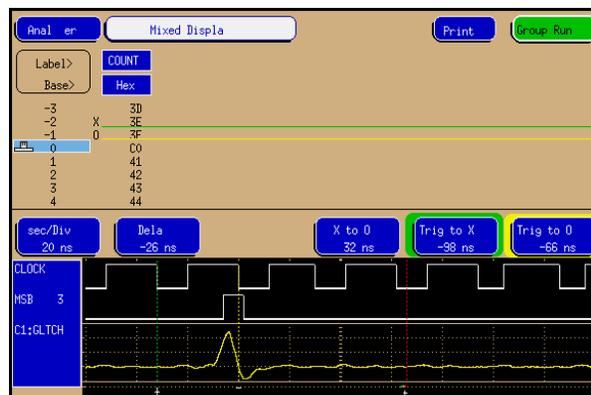
## Cross-Domain Time Correlation

Once we have successfully triggered all our measurement modules and finished data capture, we need to look at the captured data. We are all familiar with the waveform display of a scope and discussed how to present the data captured by a state or timing analyzer earlier. In order to correlate from one domain to another, it is convenient to display data from both domains on one screen. But how can we correlate between state and timing other than the trace point? Remember, the timing analyzer uses an internal sampling clock that is asynchronous to the system, while the state analyzer samples synchronously to the target system. If we count the time between the external state samples, we have enough time information to correlate from any point of the timing analyzer waveform to the appropriate location of the state analyzer listing.

## Application Example

In figure 3.2 you see the state analyzer is used to trigger on a certain memory access. Both the timing analyzer and scope are triggered by the state analyzer to provide timing information over multiple channels as well as parametric information on fewer channels. Note that the cursors are used to correlate between time domain (scope and timing analyzer) and data domain (state analyzer).

Figure 3.2 Cross-domain measurement



## How Do I Connect To My Target System?

# 4

So far we've talked about some of the differences between scopes, timing and state analyzers. Before we're ready to apply these new tools, we should talk about one more subject – the probing system.

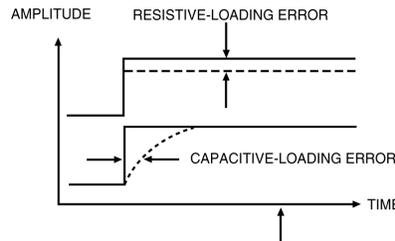
From using an oscilloscope, you're probably familiar with passive probes. A scope probe is designed to gain easy access to the target system while minimizing the signal distortion. Since we want to look at parametric information like voltage levels and rise times, it is important that the probe doesn't load the circuit under test significantly. A typical scope probe has  $1\text{ M}\Omega$  impedance shunted by  $10\text{ pF}$ , depending on the bandwidth required.

On the other hand, a logic analyzer probe is designed to allow connection of a high number of channels to the target system easily by trading off amplitude accuracy of the signal under test. Remember that a logic analyzer only distinguishes between two voltage levels! Traditionally, logic analyzers used active probe pods, which had an integrated signal detection circuitry for eight channels integrated. From these pods, we could connect with leads to the circuit under test.

The typical impedance of a logic analyzer probe is in the area of  $100\text{ k}\Omega$  shunted by  $8\text{ pF}$  at the input of the active pod. The connecting wires, however, add another  $8\text{ pF}$  stray capacitance, giving a total of  $16\text{ pF}$  per channel.

### Resistive vs. Capacitive Loading

How does the probe impedance affect my measurement? Two sources of loading, resistive and capacitive are the main cause of signal distortion. Resistive loading affects the amplitude of the output through a resistive divider effect.



Capacitive loading affects the timing of the signal under test by rounding and slewing the edges.

Amplitude errors from resistive loading are not significant enough to affect most circuit's performance, even when probing with the  $1\text{ GHz}$  scope probes with  $10\text{ k}\Omega$  resistance. In fact, most logic families can operate correctly with as much as a 10% error in amplitude. Because most of these digital ICs exhibit typical output impedance in the low hundreds of ohms or less, you can use a probe with tip resistance measuring a few  $\text{k}\Omega$ .

The capacitive loading of probes becomes more important as clock rates continue to increase in new designs. Because of this increase of clock rates, circuits are more sensitive to timing errors of even a few nanoseconds. The basic timing-error immunity, on the other hand, is limited by a circuit's clock rate. A CMOS circuit that drives a given load may operate correctly even with a higher clock rate, but the extra capacitive loading of a probe on that circuit can produce unexpected timing problems.

Table 4.1 Increases in CMOS gate delay due to probe capacitance

Capacitance	Standard CMOS Delta T	High Speed CMOS Delta T
$15\text{ pF}$	$25\text{ ns}$	$2.5\text{ ns}$
$8\text{ pF}$	$13\text{ ns}$	$1.3\text{ ns}$
$2\text{ pF}$	$3\text{ ns}$	$0.3\text{ ns}$

---

## Probing Solutions

Physical connections to digital systems for debugging must be reliable and convenient to deliver accurate data to the logic analyzer with minimum intrusion to the target system being debugged. Agilent Technologies offers a broad selection of probes and accessories for connection to target systems.

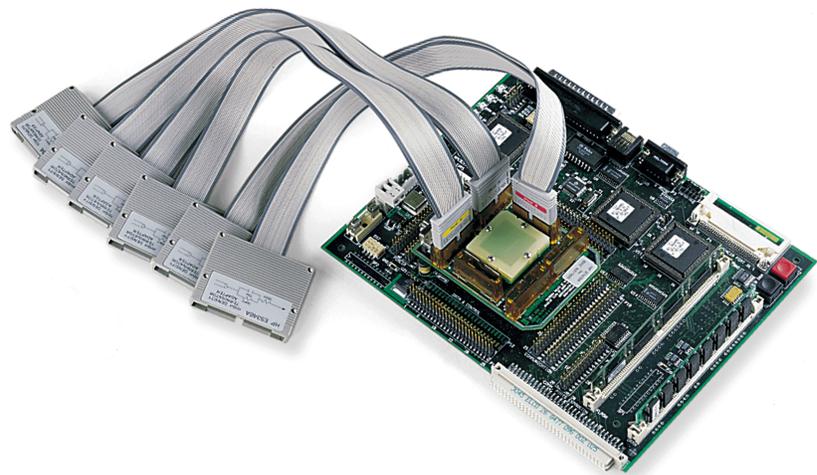
A common probing solution is the passive probe with sixteen channels per cable. Each channel is terminated at both ends with  $100\text{k}\Omega$  and  $8\text{ pF}$ . You can best compare the passive probe electrically with the scope probe. The advantage of the passive probing system, besides small size and high reliability, is that you can terminate the probe right at the point of connection to the target system. This avoids additional stray capacitance due to the wires from the larger active pods to the circuit under test. As a result, your circuit under test only "sees"  $8\text{ pF}$  load capacitance instead of  $16\text{ pF}$  with previous probing systems.

## Analysis Probe and Other Accessories

Connecting a state analyzer to a microprocessor system requires some effort in terms of mechanical connection and clock selection. Remember, we have to clock the state analyzer whenever data or addresses on the bus are valid. With some microprocessors it might be necessary to use external circuitry to decode several signals to derive the clock for the state analyzer. An analysis probe provides not only fast, reliable and correct mechanical connection to your target system, but also the necessary electrical adaptation like clocking and demultiplexing to capture your system's operation correctly.

Some microprocessors prefetch information from memory that may never get executed. Analysis probes can also distinguish prefetched information from executed. Furthermore, an analysis probe typically comes together with a disassembler to decode the hexadecimal information into microprocessor mnemonics, as discussed earlier.

Figure 4.1 An analysis probe



## Summary

---

We hope this application note has helped you gain a better understanding of what a logic analyzer is and does. Since most analyzers are made up of two major parts, timing and state, we have covered them separately. But together, they make up a powerful tool for the digital designer.

The timing analyzer is closely akin to the oscilloscope, but is better suited to bus-type structures or applications where you are dealing with many lines. It also has the ability to trigger on patterns among the lines, or even glitches.

A state analyzer is most often viewed as a software tool. In reality, it also has many uses in the hardware domain. Because it gets its clock from the system under test, it can be used to catch data when the system sees it – on the system's clock.

Armed with this fundamental knowledge, you can now use a logic analyzer with confidence.

[www.agilent.com](http://www.agilent.com)

For more about Agilent Technologies test and measurement products, applications, services, and for a current sales office listing, visit <http://www.agilent.com/find/tmdir>. You can also contact one of the following centers and ask for a test and measurement sales representative.

**United States:**

Agilent Technologies  
Test and Measurement Call Center  
P.O. Box 4026  
Englewood, Colorado 80155-4026  
(tel) 1 800 452 4844

**Canada:**

Agilent Technologies Canada Inc.  
5150 Spectrum Way  
Mississauga, Ontario  
L4W 5G1  
(tel) 1 877 894 4414

**Europe:**

Agilent Technologies  
Test & Measurement  
European Marketing Organization  
P.O. Box 999  
1180 AZ Amstelveen  
The Netherlands  
(tel) (31 20) 547 9999

**Japan:**

Agilent Technologies Japan Ltd.  
Call Center  
9-1, Takakura-Cho, Hachioji-Shi,  
Tokyo 192-8510, Japan  
(tel) (81) 426 56 7832  
(fax) (81) 426 56 7840

**Latin America:**

Agilent Technologies  
Latin American Region Headquarters  
5200 Blue Lagoon Drive, Suite #950  
Miami, Florida 33126  
U.S.A.  
(tel) (305) 267-4245  
(fax) (305) 267-4286

**Australia/New Zealand:**

Agilent Technologies Australia Pty Ltd.  
347 Burwood Highway  
Forest Hill, Victoria 3131  
(tel) 1 800 629 485 (Australia)  
(fax) (61 3) 9272 0749  
(tel) 0 800 738 378 (New Zealand)  
(fax) (64 4) 802 6881

**Asia Pacific:**

Agilent Technnologies  
24/F, Cityplaza One, 1111 King's Road,  
Taikoo Shing, Hong Kong  
(tel) (852) 3197 7777  
(fax) (852) 2506 9284

Technical datais subject to change  
5968-8291E 11/99



**Agilent Technologies**  
Innovating the HP Way