



Agilent Technologies

Digital Hardware Debug Techniques

September 26, 2001

presented by:

Judith Smith

Agenda

- **Digital system development process**
- **Digital debug methodology overview**
- **Applying logic analyzers to digital debug**
- **Measurement challenges**
 - **Finding the cause of a system crash**
 - **Finding the cause of data corruption**
- **Test equipment considerations**

This presentation covers digital hardware debug techniques that save you time and help you solve critical debug challenges. The majority of the presentation will cover the solution for the following measurement challenges:

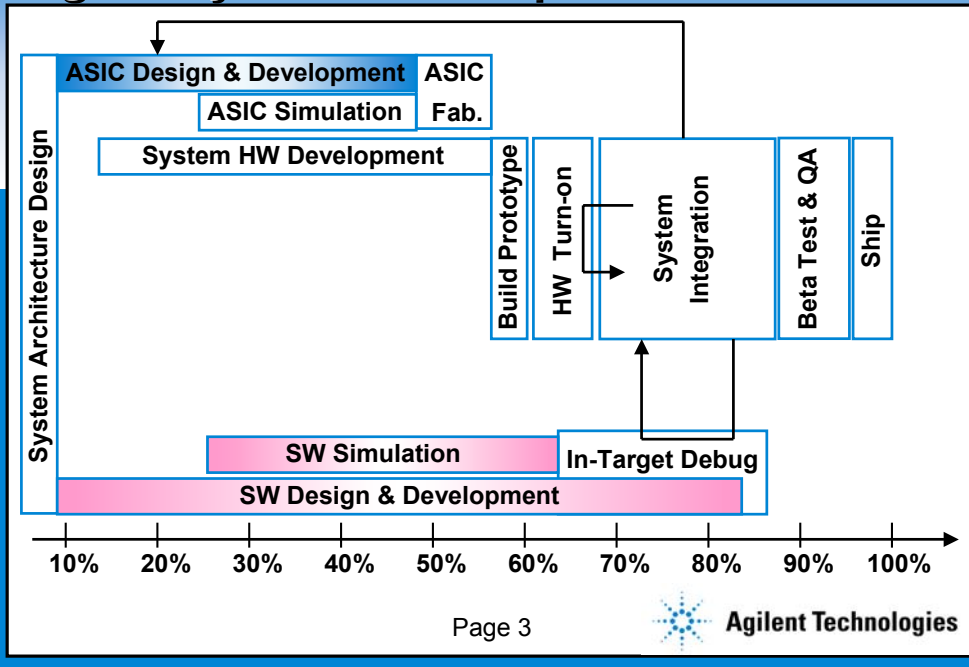
- **Finding the cause of a system crash**
- **and finding the cause of data corruption**

We'll start off with a quick overview of:

- **the digital system development process**
- **the methodology for digital hardware debug**
- **and how logic analyzers help you verify system operation and identify problems**

We'll close with things to consider in your debug tool selection in order to meet your application and measurement needs.

Digital System Development Process

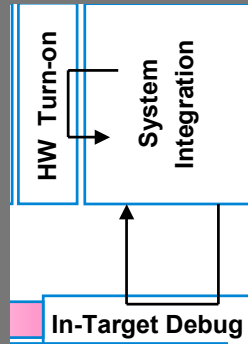


Developing a digital device can involve many people, tasks, and stages. From the initial system architecture design to the first shipment of your product, you have to resolve numerous issues in order to complete your project on time.

Each stage presents its own unique set of challenges. While many tasks can be worked in parallel, others occur in serial and are dependent on other tasks being complete.

Agilent provides tools and measurement assistance in each stage of the development process. Agilent's design and debug tools get you to market faster and easier.

Digital System Development Process



There are multiple topics that could be addressed but for today's presentation we will focus on solutions to some of the digital hardware debug challenges you encounter during board turn-on and in-target debug.

So, we've seen an overview of the product development cycle, but what does the world look like from your view as a digital hardware designer?

Engineer's Dilemma



**If you knew what the problem was,
you wouldn't have to look for it.**

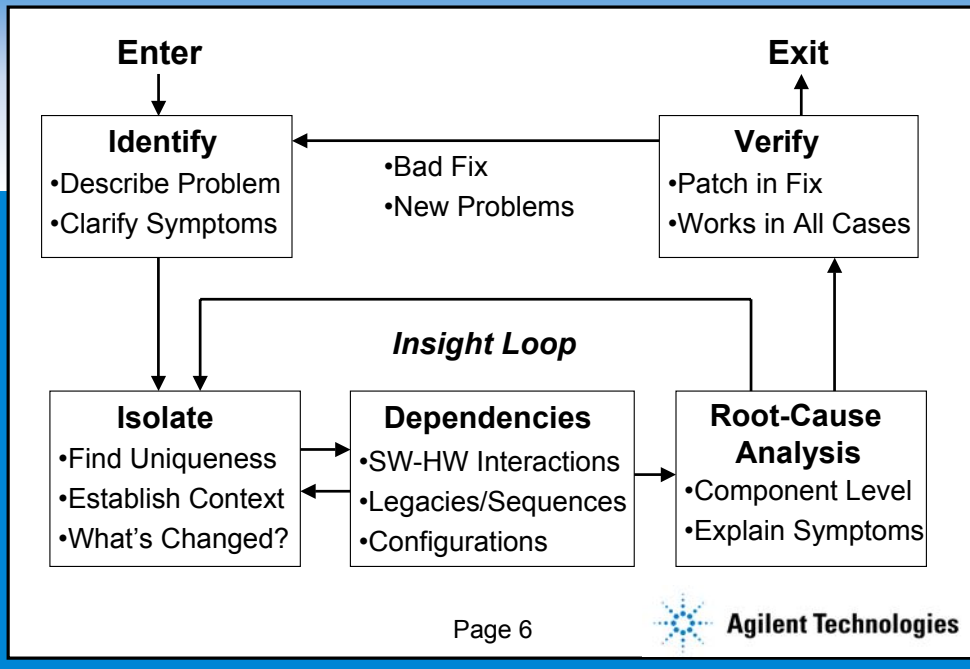
**How do you capture something
when you don't know what you're
looking for?**



You are faced with multiple design decisions on a daily basis. You also encounter system problems that take time to find, let alone resolve. If you knew what was causing problems in your digital circuit, you wouldn't have to look for it. You'd fix it and move on. Your specialty is designing solutions.

Agilent's specialty is helping you with tools and techniques to identify your problems quickly so that you can spend your time designing solutions.

Digital Debug Methodology Overview



During product development, you need to take advantage of everything that allows you to hit market windows and beat your competitors to market. A big part of reducing time to market is reducing the time it takes to identify, isolate and solve problems that show up during board turn-on and in-target debug.

The insight loop is an iterative troubleshooting cycle that includes:

1. Identifying the error
2. Isolating and understanding the problem
3. Performing root-cause analysis down to the component level to explain symptoms.

Getting to root cause is essential in any debug situation and a logic analyzer is an essential tool because it provides visibility into system operation. Without it, you run the risk of designing a misguided fix based on a faulty diagnosis - a mistake that can create even more problems than it solves.

Why You Need a Logic Analyzer

- **See many signals at once**
- **Gain insight into digital circuit operation**
- **View signals same way the hardware does**
- **Trigger on edges, patterns and complex sequences of events to capture system activity**

A logic analyzer is an essential tool for digital debug because of its multiple measurement modes, triggering and data storage capabilities.

You should use a logic analyzer when you need to :

- **See many signals at once.**
- **Look at signals in your system the same way your hardware does.**
- **Trigger on a pattern of highs and lows on several lines and see the results.**

Measurement Challenges

Example #1

Finding the Cause of a System Crash

Page 8



In the following examples we will review two digital measurement challenges and how to solve them using a logic analyzer. In each example we will provide an overview of the solution, as well as go into each step in depth, providing not only the reasoning for each step but the way to do it.

The first digital measurement challenge we will cover is finding the cause of a system crash.

System Crash

Why Are System Crashes So Difficult?

- Only visible symptom is the crash itself
- Occurs intermittently, from seconds to days
- There are no live signals to trigger on



Invariably, every designer encounters a system that has elusive system crashes. The difficulty in getting to the root cause of a crash is not the analysis of events leading up to the crash, but rather the ability to even capture a system crash for analysis. So why is it so difficult to capture a system crash?

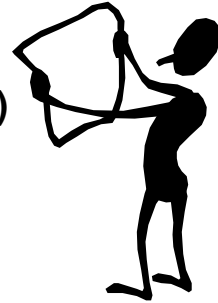
Typically,

- The only visible symptom is the crash itself
- Crashes occur intermittently, anywhere from seconds to days until the next occurrence
- When a crash occurs there are no live signals available to trigger the logic analyzer.

System Crash

How Do You Find the Problem?

- See activity leading up to crash
- Determine how to see activity
 - System execution by state (state)
 - High level of resolution (timing)
- Select corresponding method for capturing pre-crash activity



So, what do you need in order to analyze a crash and get to the root cause of the problem? It's critical to see the activity leading up to the crash. The important clues are most likely found there.

It is also important to determine just what type of view you want - either system execution, event by event, or high resolution timing traces.

System Crash - State Solution

State Method for Capturing Pre-crash Activity

1. Define a trigger event that will never occur
2. Position trigger at end of acquisition memory
3. Acquire system activity continuously
4. Stop measurement manually after crash

If you want to see what the system is executing, use state mode to capture system activity.




The major steps for capturing pre-crash activity using state mode are:

1. Define a trigger event that will never occur
2. Position the trigger at the end of acquisition memory
3. Acquire system activity continuously
4. Stop the measurement manually after the system crashes

System Crash - State Solution

1. Define a Trigger Event That Will Never Occur

- Why?
 - Keep the analyzer acquiring samples
- How?
 - Create/select label with a hard-wired bit
 - Set trigger to opposite of hard-wired level

Bus/Signal Name	Channels Assigned	Pod 2															
		Threshold: TTL (1.50 V)															
		↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↓	↓
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
 No Trigger	Pod 2[0]																✓
 ADDR	Pod 1[15:0]																
 DATA	Pod 2[15:4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Page 12



Let's look at each step in greater detail and review not only why the step is important, but also how to make it happen.

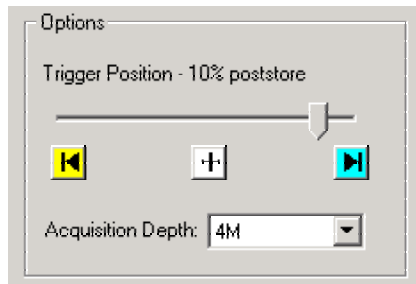
The first step is to define a trigger event that will never occur so that the logic analyzer will continue to acquire data until manually told to stop.

To do this, create a new label or select an existing label in the logic analyzer setup menu. Here we show a label called 'No Trigger' that represents the signal connected to Bit 0 of Pod 2. The signal is hard-wired to ground. In the trigger specification, this new label is used to set up the trigger event. The trigger event is defined as the occurrence of the opposite level of the given signal, i.e. high for a TTL signal hard wired to ground, which will never occur.

System Crash - State Solution

2. Position Trigger at End of Acquisition Memory

- **Why?**
 - See all activity leading up to crash/trigger
- **How?**
 - Use analyzer's adjustable trigger position

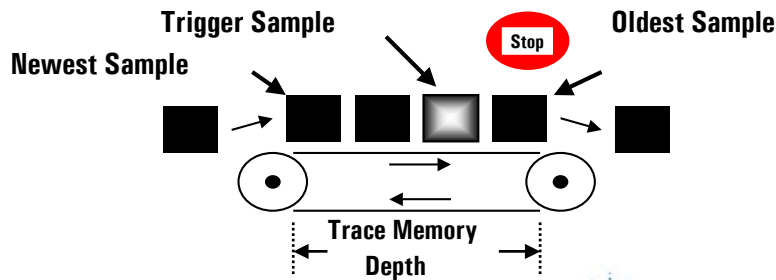


The second step is to position the trigger at the end of acquisition memory so that you can see all of the activity leading up to the crash. You can adjust the logic analyzer's trigger position to be placed anywhere within the acquisition memory. In this instance, 90% of the acquisition memory contains the events leading up to the trigger event, and 10% of the acquisition memory is used to store the events after the trigger event.

System Crash - State Solution

3. Acquire System Activity Continuously

- Why?
 - Need pre-crash activity for analysis
- How?
 - Analyzer overwrites older data with latest



Page 14

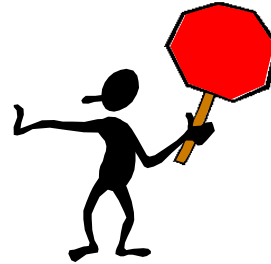
The third step is to start the logic analyzer measurement and continue to acquire system activity. In state mode, the analyzer uses a signal from the device under test as the sampling clock. As long as the target is alive, the analyzer will continue to sample and store system activity into the logic analyzer's memory.

The analyzer's memory can store a given number of samples. Whenever a new sample is acquired, the oldest sample currently in memory is thrown away if the memory is full. In this manner, the logic analyzer continues to store only the most recent activity.

System Crash - State Solution

4. Stop Measurement Manually after Crash

- **Why?**
 - To see what has been stored to memory
- **How?**
 - Press the instrument's Stop key



The final step is to manually stop the measurement once you see the system crash. When a crash occurs, the signals used to sample data into logic analyzer memory go away, therefore no additional samples are acquired or stored by the logic analyzer. Pressing the logic analyzer's Stop key creates an artificial trigger. The logic analyzer has the most recent activity prior to the crash stored in memory.

You now have the critical data needed to determine the cause of the crash.

System Crash - State Solution

Technique Learned

- **Creating an artificial trigger**
 - **Set up a label for a signal hard-wired to ground**
 - **Specify trigger event as opposite value of the new label**
 - **Remember signal used for analyzer's state clock stops when system crashes**
 - **Stop the measurement manually**



From this example you've learned how to create an artificial trigger:

- **Set up a label for a signal that is hard-wired to ground**
- **Specify the trigger event as the opposite value of the new label**
- **Remember signal used for analyzer's state clock stops when system crashes**
- **Stop the measurement manually**

System Crash - Timing Solution

Timing Method for Capturing Pre-crash Activity

1. Find a periodic/consistent signal in the system
2. Determine maximum time interval for signal
3. Position trigger at end of acquisition memory
4. Set trigger to occur if time interval is exceeded

If you want to see system timing information with high resolution, use timing mode to capture system activity.

The major steps for capturing pre-crash activity using timing mode are:

1. Find a periodic or consistent signal in the system
2. Determine the the maximum time interval for the signal
3. Position the trigger at the end of acquisition memory
4. Set the trigger to occur if the time interval is exceeded

System Crash - Timing Solution

1. Find a Periodic/Consistent Signal in System

- **Why?**
 - **Absence of predictable signal is trigger**
- **How?**
 - **Use predictable signal like address strobe**

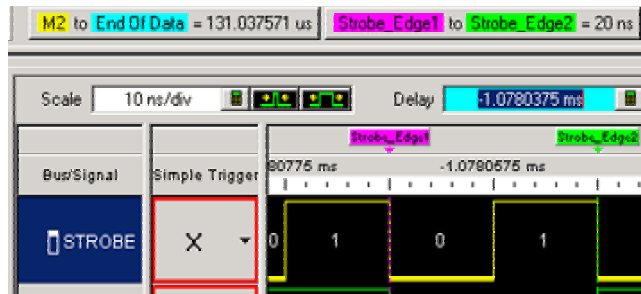


The first step is to find a signal in the system that is periodic or consistent - like an address strobe or a periodic interrupt. The absence of this signal indicates that the system has crashed.

System Crash - Timing Solution

2. Determine Time Interval for Signal

- Why?
 - Need interval value to specify trigger
- How?
 - Use analyzer marker measurement

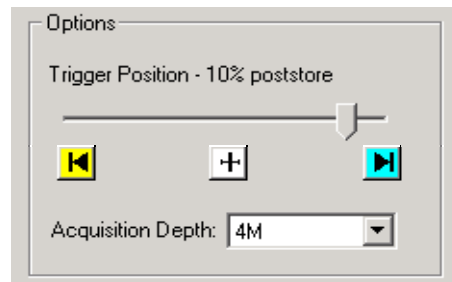


Next, determine the maximum time interval for the chosen signal. The easiest way is to take a logic analyzer trace and use the markers to measure the time interval for the signal.

System Crash - Timing Solution

3. Position Trigger at End of Acquisition Memory

- **Why?**
 - See all activity leading up to crash/trigger
- **How?**
 - Use analyzer's adjustable trigger position



Next, position the trigger at the end of acquisition memory so that you can see all of the activity leading up to the crash. Remember, you can adjust the logic analyzer's trigger position to be placed anywhere within the acquisition memory. In this instance, 90% of the acquisition memory contains the events leading up to the trigger event, and 10% of the acquisition memory is used to store the events after the trigger event.

System Crash - Timing Solution

Technique Learned

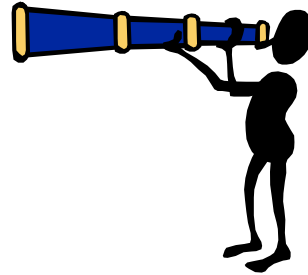
- Use the absence of a periodic signal as the trigger event for a system crash



From this example you've learned how to use the absence of a periodic signal as the trigger event for a system crash.

Taking the Crash to Root Cause

- Review trace for clues
- Use clues to:
 - Determine the cause from the existing trace
 - Decide what needs to be acquired next



Typically, the difficulty in getting to the root cause of a crash is not the analysis of events leading up to the crash, but rather the ability to even capture a system crash for analysis. Once you've captured the events leading up to the crash, you have information that will lead you to the cause of the crash.

Review the trace for clues. In many cases the trace you've captured will provide a good indication of the cause of the crash. If not, the clues will help you decide what needs to be acquired next to give you the necessary information.

Q & A

Measurement Challenges

Example #2

Finding the Cause of Data Corruption

Tracking Symptom to Root Cause

Page 25



The second digital measurement challenge we will cover is finding the cause of data corruption. In each example we will provide an overview of the solution, as well as go into each step in depth, providing not only the reasoning for each step but the way to do it.

Data Corruption

Why Is Data Corruption So Difficult?

- Symptom may be totally unrelated to cause
- Cause could be software, hardware or interaction of the two
- Wide variety of causes may require cross-domain analysis



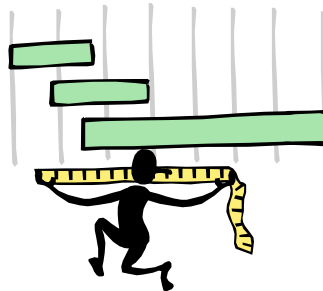
In a digital system, sometimes the symptom of a problem may be totally unrelated to the cause. The cause for a symptom may be software, hardware or the interaction of the two. The symptom may be more easily found using one measurement domain while the cause of a problem is best captured in a different measurement domain.

Cross-domain measurements let you combine the logic analyzer's different measurement capabilities to solve hard-to-find problems in your digital system.

Data Corruption

Measurement Modes

- **State**
 - Synchronous sampling
- **Timing**
 - Asynchronous sampling
- **Oscilloscope**
 - Voltage resolution and parametric measurements



State analysis contributes synchronous acquisition and software analysis capability.

Timing analysis adds high-resolution asynchronous acquisition and control, as well as bus signal analysis capability.

Analog (oscilloscope) analysis provides voltage resolution and parametric analysis measurements.

Data Corruption

How Do You Find the Problem?

- Identify the symptom
- Capture activity related to the symptom
- Use multiple measurement modes to uncover the root cause



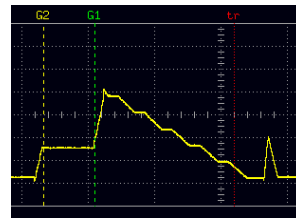
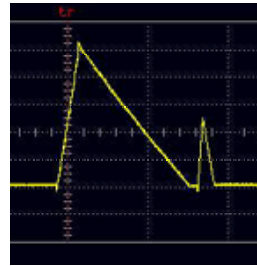
So how do you find the cause of data corruption?

- 1. Use the symptom of the problem as the main triggering event.**
- 2. Isolate the area that is causing a symptom.**
- 3. Use all measurement domains possible to identify the root cause.**

Data Corruption - Incorrect D/A Output

Identify the Symptom

- D/A output should be triangular waveform
- Scope shows distortion in waveform
- Distortion varies over time



Our first data corruption example is a D/A that is generating an incorrect waveform. The target is supposed to produce a triangular waveform output signal. When looking at the D/A output with an oscilloscope you see a distortion in the expected triangular waveform. By running the scope repetitively you see that the distortion also varies over time.

Data Corruption - Incorrect D/A Output

Method for Determining D/A Output Corruption

1. Use scope trigger to arm logic analyzer
2. Analyze activity when waveform generated
3. Verify expected cause



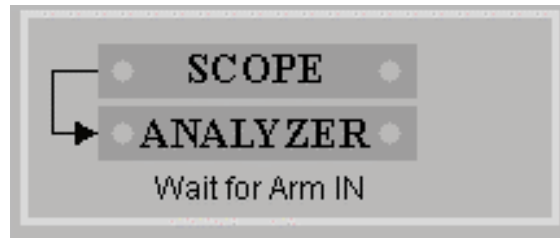
The major steps for determining the cause of the D/A output corruption are:

1. Use scope trigger to arm logic analyzer
2. Analyze activity when waveform generated
3. Verify expected cause

Data Corruption - Incorrect D/A Output

1. Use Scope Trigger to Arm Logic Analyzer

- Why?
 - Correlate measurement activity
- How?
 - Arm logic analyzer when scope triggers



To correlate measurement activity, arm the logic analyzer to trigger when the oscilloscope triggers. This allows you to determine the relationship between different parts of the system at the same moment in time.

Data Corruption - Incorrect D/A Output

2. Analyze Activity When Waveform Generated

- Why?
 - See what else happens at the same time
- How?
 - Acquire processor activity when generating waveform

PC	MPC821/860 Inverse Assembler
Symbols	10=hex, 10.=decimal, %10=binary
/q,elf;reset+0430	mtmr r26
/q,elf;reset+0434	subi r1 r1 0008
/q,elf;reset+0438	bl q.:isr;ext_exception
isr;ext_exception	mfscr r0 lr

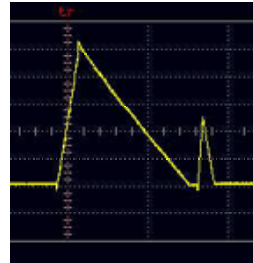
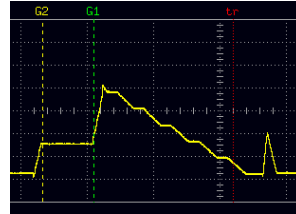
It is important to see what else is happening in the system when the waveform distortion occurs. The logic analyzer is used to acquire processor activity when the D/A output distortion occurs.

By examining the processor mnemonics we learn that the defects in the waveform occur when interrupt service routines execute during the generation of the waveform.

Data Corruption - Incorrect D/A Output

3. Verify Expected Cause

- Why?
 - Ensure you get to root cause
- How?
 - Enable/disable interrupt



To fully verify that the interrupt service routines are the culprit, you can use an emulation probe to enable and disable the suspect interrupt line. When turned on, it causes the D/A output distortion. When turned off, the D/A produces the correct triangular waveform output.

Data Corruption - Display Error

Identify the Symptom

- **System writes test pattern to display**
- **Incorrect character intermittently written to display**



Let's look at another example of data corruption and the use of cross domain analysis to determine the cause. In this example the system writes a test pattern of block characters to the display. Intermittently, an incorrect character is written to the display.

Data Corruption - Display Error

Method for Determining Cause of Display Error

1. Capture writes to display (state mode)
2. Review timing relationships (timing mode)
3. Examine signal characteristics (scope)

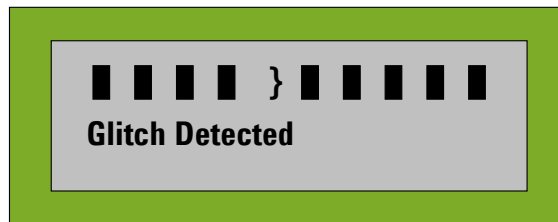
The major steps for determining the cause of the display error are:

1. Capture writes to the display using state mode
2. Review timing relationships using timing mode
3. Examine the signal characteristics using a scope

Data Corruption - Display Error

1. Capture Writes to Display (State Mode)

- Why?
 - To capture activity surrounding error
- How?
 - Set trigger for write of non-block character



The state analyzer is an excellent tool for monitoring accesses to a memory location. You can set up the analyzer to capture all reads and writes to a specific address and trigger if an unexpected value occurs. After tracing all accesses to the problem address, you can quickly determine if any unexpected values were written by examining the state listing.

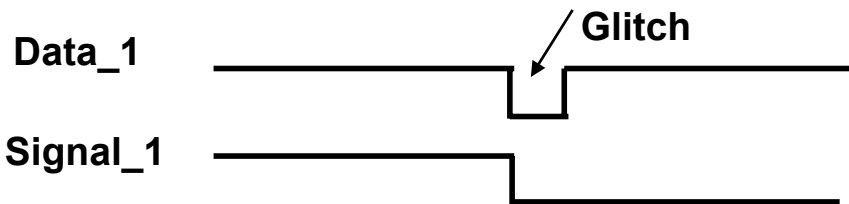
In this example, the logic analyzer is set to trigger when a non-block character is written to the display. Block characters were being sent to the display before the error occurred.

If the software had been writing the incorrect value, you could examine the state listing to see which part of the program was executing at that time. If only correct values are written, the problem might be in the hardware.

Data Corruption - Display Error

2. Review Timing Relationships (Timing Mode)

- Why?
 - Uncover hardware timing problems
- How?
 - Trigger on write error, examine relationships



Looking at the same reads and writes with the timing analyzer gives you more insight as to the possible cause of the problem. Looking at the timing relationships between the address, data, and control lines, you can uncover hardware problems such as timing violations, race conditions, or improper control sequences.

Occasionally, looking at signals with the timing analyzer does not uncover the cause of the problem. If you have an overdrive problem, the timing analyzer might not detect it. Perhaps you do see that one of the data bits is low when it should be high, or that a control signal has a glitch. But you still don't know the cause of the problem. To understand the cause of these parametric types of problems, you can use a scope to look at the analog domain.

In this case, every time we see a glitch on Data_1, Signal_1 is transitioning from high to low.

Data Corruption - Display Error

3. Examine Signal Characteristics (Scope)

- **Why?**
 - To understand parametric problems
- **How?**
 - Arm the scope from the timing analyzer



Analyzing the analog properties of your digital signals helps you uncover problems such as inadequate voltage swings, cross-talk, and ground bounce.

When the non-block character occurs, the offending signal is being pulled from a high to low when another signal goes from high to low. Cross domain measurement techniques identify energy being cross-coupled between two traces that are routed too closely together.

Data Corruption

Technique Learned

- **Cross-domain analysis methodology**
 - Use the symptom of the problem as the main triggering event
 - Use other measurement domains to isolate area causing the symptom
 - Use additional measurement domains to identify root cause



From these two examples you've learned how to use cross domain analysis to get to the root cause of the problem.

- Use the symptom of the problem as the main triggering event
- Use other measurement domains to isolate the area causing the symptom
- Use additional measurement domains to identify root cause

Data Corruption - Write to Display

Use of Multiple Measurement Modes

- State analysis - to show how it happened
- Timing analysis - to show when it happened
- Oscilloscope - to show why it happened

Capturing complex failures often requires state analysis to show how it happened, timing analysis to show when it happened and oscilloscope analysis to show why it happened.

Test Equipment Considerations

- What measurements are you trying to make?
- How many signals will you be analyzing?
- How much memory do you need?
- What speed requirements do you have?
- Have you planned how to probe your system?
- What processors and buses are you using?
- Do you need to control the processor?

To determine your test equipment needs, consider the following with regard to your target and application:

- What measurements are you trying to make?
- How many signals will you be analyzing?
- How much memory do you need?
- What speed requirements do you have?
- Have you planned how to probe your system?
- What processors and buses are you using?
- Do you need to control the processor?

Conclusion

Q & A

Page 42



Agilent Technologies

We hope this presentation has helped you gain a better understanding of digital hardware debug techniques.