Fundamentals of Timing Analysis

Tektronix
Enabling Innovation

Table of Contents

Introduction

Digital circuits are designed to work in a logical and organized manner, with events occurring in a well-timed order. Timing analysis, a fundamental aspect of digital system validation and debugging, is required to ensure this order commences properly and the digital circuit performs as expected. It is often a starting point for hardware engineers to verify functionality and identify problems in their system.

Although validating system timing may sound simple, the process of measuring and calculating all applicable parameters can be tedious and time consuming. It is essential that digital design engineers understand the challenges – both existing and emergent – they will face as they validate and debug their systems.

This primer highlights important timing analysis issues, such as common errors, their causes and ways to efficiently confront them. The latest tools for capturing elusive timing errors and expediting system verification will also be discussed.

Timing Analysis Challenges

To operate properly, digital circuits must run at a certain predefined pace, sequence and specification. Any violation of these predefined criteria impacts system timing and can cause problems or errors. Therefore, timing analysis is often used as a starting point to debug a system and find the root cause of known problems.

Recent trends in the digital design realm have created new challenges for design engineers. High-speed buses render digital systems more sensitive and vulnerable to timing errors. The popularity of Field-Programmable Gate Array (FPGA) chips has made it difficult to correlate internal logic with external signals. And with the complexity of today's systems, timing errors are often elusive, necessitating significant effort to identify and resolve.

Complex System Errors

Most high-speed digital systems typically use some type of internal communication mechanism to transmit data between sub-systems. A data frame is generated and sent from one subsystem, and then travels through several "blocks" to arrive at its destination. Oftentimes these links do not function reliably, sporadically failing to send the right signals on time and causing the system to crash. It is the test engineer's responsibility to find out why. These engineers typically face a complex system with several subsystems and hundreds of channels.

The questions then become: how does the engineer correlate signal activities and find clues that help resolve the problem? If there are signals running inside and outside a FPGA, how does the engineer attain visibility inside the FPGA? If the engineer suspects a timing error, what tools can help effectively capture the timing error, which may only cover several nanoseconds, amidst complex system activities?

System Metastability

Every engineer has experienced scenarios where errors are identified with no clues as to their root cause, especially in the early stages of the validation process. For example, many embryonic systems perform inconsistently. Sometimes they crash; other times they fail to deliver expected outputs. There are clearly "bugs" hiding somewhere in the system, but the cause can be almost anything. After running the validation board many times, the engineer may find that there is a correlation between the problem and system software trying to write certain values to a control bus. On the clock's rising edge, the value of the control bus may be going through D flip-flops and written to a certain address in the system's memory.

To validate those flip-flops, the engineer employs an oscilloscope to validate channels, one-by-one, to determine whether there are setup/hold violations occurring on the D flip-flops. The violation may be caused by an unexpected propagation delay, meaning the signal trace propagation delay takes more time than expected and may violate the setup/hold specification of the D flip-flop, causing system failure.

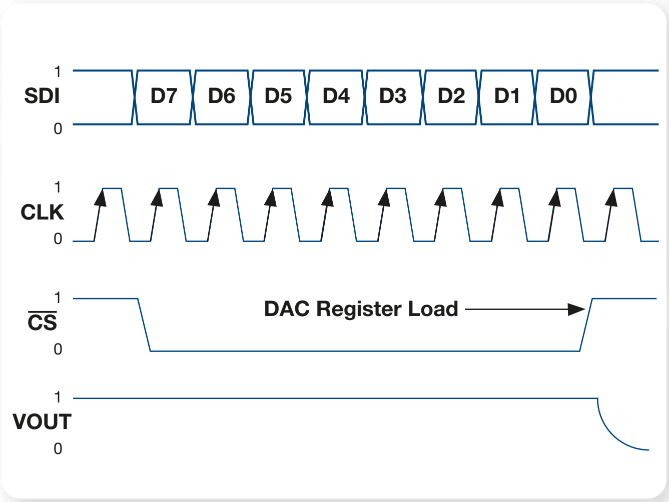
Nonetheless, has the engineer captured the entire setup/hold violation? Do all the other flip-flops in the system work properly? Setting up oscilloscope probing for every channel is a very time consuming process. Are there better tools to use when capturing these violations in a complex system with so many waveform activities?

Linking Analog and Digital Domains

Many engineers must design embedded systems with digital potentiometers, a group of serial digital-to-analog converters (DACs) with a serial peripheral interface (SPI). They have a FPGA sending data and clock to these serial DACs. To ensure the DAC clock is aligned to the correct data, a component datasheet specifies the clock counts and data (see Figure 1). When a chip selects a signal, every clock edge will log the data into the DAC.

The analog output often indicates that errors have occurred, and the engineer must determine the cause. However, how can the engineer correlate the digital signal inputs with the analog outputs? An oscilloscope may have captured glitches on the signal traces, but are they responsible for the errors? Are there better tools that enable the engineer to trigger on the glitch and put all the analog and digital waveforms in a single screen?

The aforementioned cases unveil intrinsic characteristics of timing errors, which can be elusive to capture and harmful to the system. To make matters worse, the same timing errors can appear in several different forms, making it even more difficult to identify and troubleshoot them.



▶ **Figure 1.** *Typical DAC timing diagram.*

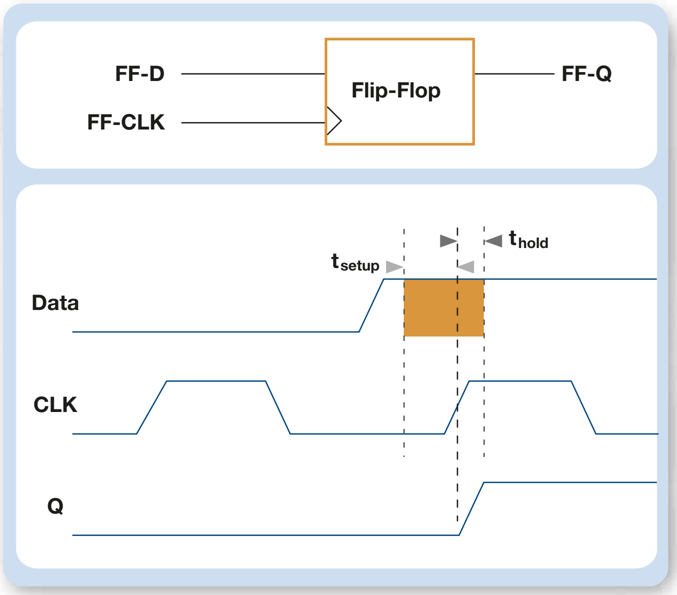
To capture the real root causes of timing errors, engineers need to have a solid understanding of their system, a good problem-solving approach and the right tools.

Common Timing Errors

In general, timing errors are those related to waveform edge positions. They take on many forms, but there several distinct commonalities.

Metastability Caused by Setup/Hold Violations

Setup Time and Hold Time are two important timing parameters for a digital design. They outline a window of time for when data must remain stable to guarantee predictable performance over the full range of operating conditions and manufacturing tolerances. More specifically, they dictate the timing requirements of the data input for a flip-flop or register with respect to the clock input.



▶ **Figure 2.** *D flip-flop diagram.*

A D flip-flop, a widely used device for synchronizing a system where the clock provides timing to the circuit, is an appropriate example (see Figure 2).

t_{setup} is the setup time required by this D flip-flop, defining the minimum time window the D input signal must be valid and stable before the clock edge. thold is the hold time required by this D flip-flop, defining the minimum time window the D input signal must be maintained to be stable after the clock edge.

When the D input signal changes within the time windows that t_{setup} and thold define, a setup/hold violation may occur. Setup/hold violations can cause a metastable Q output, which means the output of the D flip-flop is unpredictable. Metastability can cause chaos in a synchronous digital system. For example, a downstream device may become stuck at mid-voltage, rendering it extremely difficult to determine its state. Things can get worse if the system has a metastable output that drives two distinct circuits, with system logic completely failing if one circuit perceives it as “0” and the other thinks it is “1.”

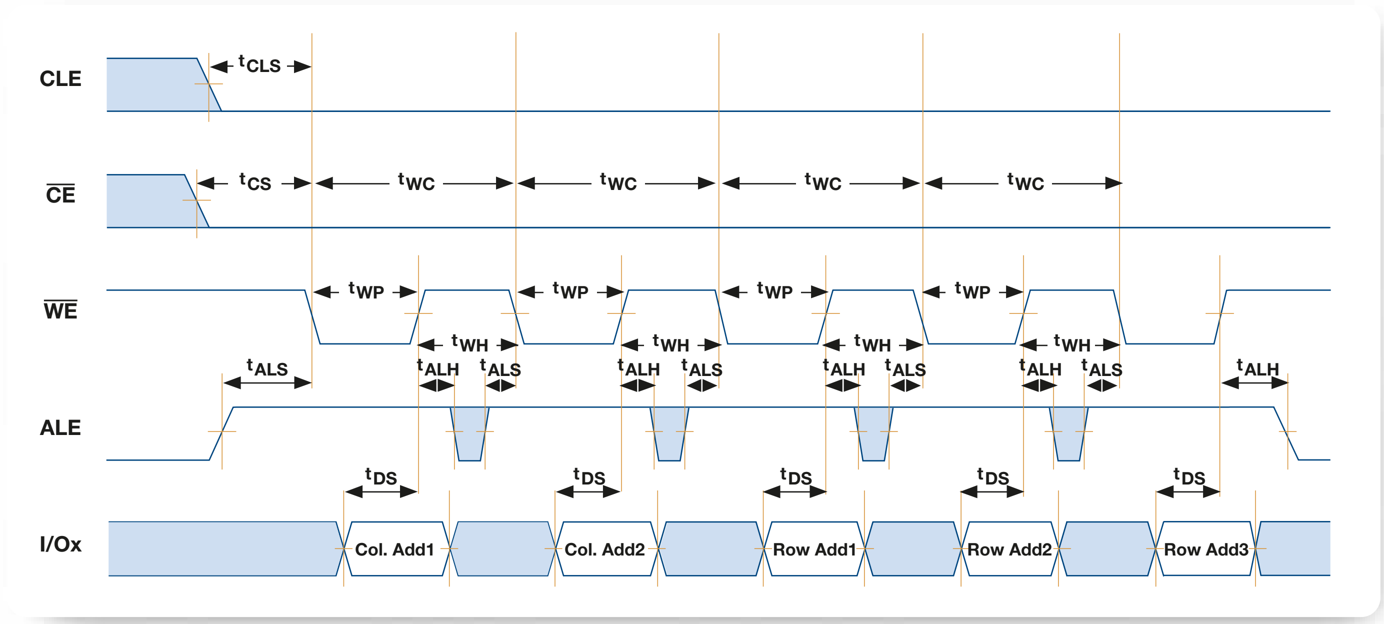
Setup and hold violations are a common headache when verifying today’s designs, including FPGAs, memory and others. And due to the complexity of these designs, it is difficult to completely eliminate this type of timing error. Therefore, it is imperative for engineers to have tools capable of capturing these errors, from a variety of channels.

An oscilloscope is a useful tool to characterize and analyze setup/hold times. However, an oscilloscope is capable of monitoring only a small number of channels. Using only an oscilloscope to capture a setup/hold violation from many channels can be very time-consuming. With many channel counts, long record lengths and powerful logic trigger capability, which triggers and marks the setup/hold timing error for the user, logic analyzers are typically used to capture setup/hold violations from multiple signal traces.

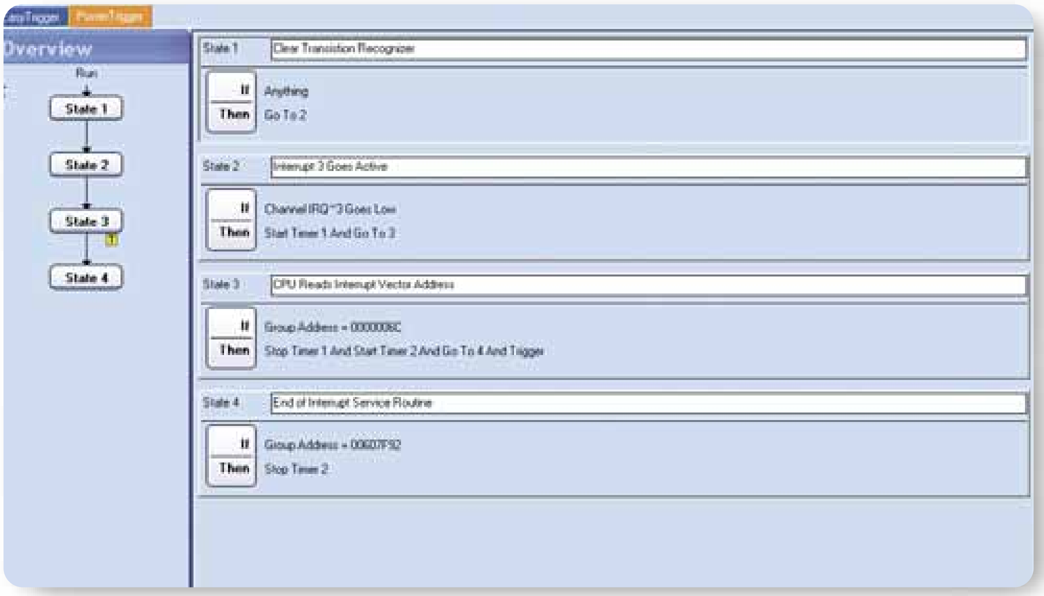
Sequencing Errors

Digital circuitry is a very organized environment where certain tasks are achieved only when specific events occur in an orderly fashion. For example, to latch the address of a NAND flash memory, a specific sequence must be followed. Digital component datasheets provide a blueprint for this and other tasks, using waveforms to illustrate the signal sequences and specify the time between signals (see Figure 3).

In the system verification phase, engineers need to spend sufficient time verifying whether signals on the circuit comply with certain requirements and specifications. If those sequences and specifications are violated, a sequence error can occur and result in system failure. For instance, a miscalculated trace length in an embedded system memory or storage circuit can lead to extra propagation delay, which may cause timing errors such as a race condition. In this example, if a race condition occurs when read and write timing sequences are violated, the engineer can possibly read incorrect data from memory, overwrite old data while it is being read or even fail to read or write any data. All of these possibilities can result in an illegal operation alert, program shutdown or even system crash.



▶ **Figure 3.** *Flash memory waveform sequence and specification example.*



State 1 Clear Transition Recognizer
If Anything
Then Go To 2

State 2 Interrupt 3 Goes Active
If Channel IRQ~3 Goes Low
Then Start Timer 1 And Go To 3

State 3 CPU Reads Interrupt Vector Address
If Group Address = 0000008C
Then Stop Timer 1 And Start Timer 2 And Go To 4 And Trigger

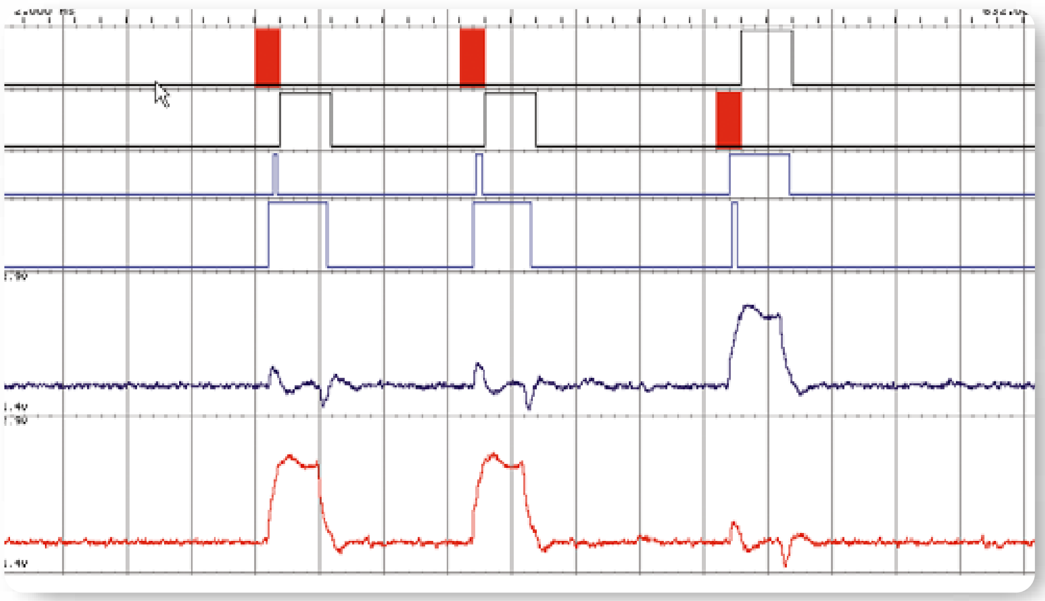
State 4 End of Interrupt Service Routine
If Group Address = 00607F92
Then Stop Timer 2

▶ **Figure 4.** *Logic analyzer power trigger example.*

Other factors can also lead to a sequence error, such as logic errors, glitches and software bugs. The biggest challenges for design engineers are analyzing timing sequences and capturing sequence errors from multiple channels amidst complex system operations.

To conquer these challenges, engineers need to design their circuitry with total system visibility during the design phase, which aids the subsequent validation and debugging phases. With system visibility built into the design, powerful tools such as logic analyzers and oscilloscopes will be able to help quickly and thoroughly test and analyze system activities.

Logic analyzers offer sophisticated trigger capabilities, which enable the user to trigger on specific events or signal sequences (see Figure 4). These capabilities enable engineers to correlate signal activities from multiple buses and expedite system validation and debugging. In addition, the logic analyzer's automatic measurement features help the user easily attain necessary timing information, such as pulse counts and delta time between edges.



▶ **Figure 5.** *A glitch created by crosstalk, which then causes a timing error.*

Jitter is another big problem for system reliability and can be difficult to analyze. Conceptually, jitter is the deviation of timing edges from their "correct" locations. For example, jitter in a memory system may cause the data line to be clocked in memory at the wrong time, which can cause system failure. This jitter could be the result of the frequency of the switching power supply introduced to the phase-locked loop (PLL); it can also be caused by an unstable clock-recovery PLL design.

The timing errors caused by signal integrity problems are often characterized by their elusive and transient appearances. Using the latest test and measurement tools, such as cutting edge logic analyzers and oscilloscopes, can effectively enhance an engineer's ability to capture and characterize these errors.

Signal Integrity Flaws in the Timing Domain

Today's digital systems are more complex and sensitive than ever before. They operate at much higher speeds, now commonly in the Gigahertz range. Serial buses are quickly replacing traditional parallel buses and breaking speed barriers in the process. In addition, fast edge logic families are being widely adopted. Because of these developments, signal traces act not only as resistors, but also capacitors and inductors, which can cause signal displacement and create serious timing errors.

At the same time, high-speed systems are increasingly sensitive, leading to signal integrity issues. Glitches and timing jitter are two examples of how signal integrity problems can cause timing errors.

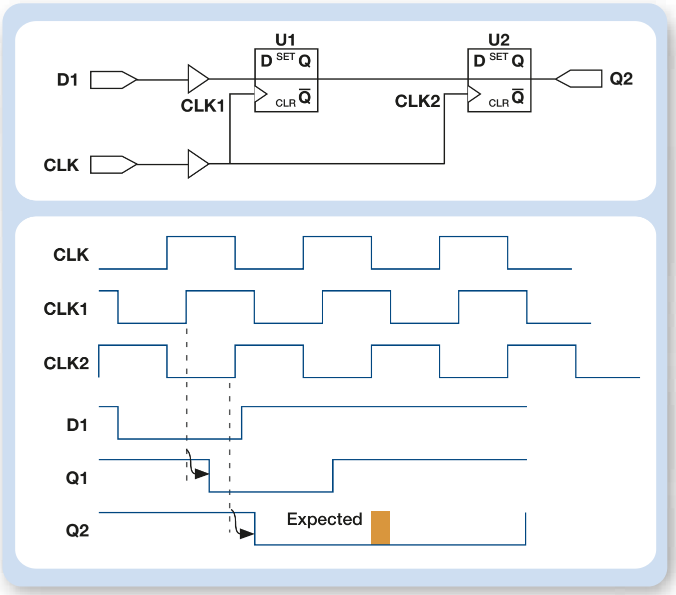
A glitch that can be ignored in a low-speed system may cause significant timing problems in a high-speed environment. These unexpected narrow pulses in the signal trace, which may or may not be interpreted as logic changes by a system, can be caused by a variety of reasons, such as race conditions, termination errors, driver errors and crosstalk. For example, if two long parallel signal traces are running close to each other, crosstalk may occur and create glitches (see Figure 5).

Causes of Timing Errors

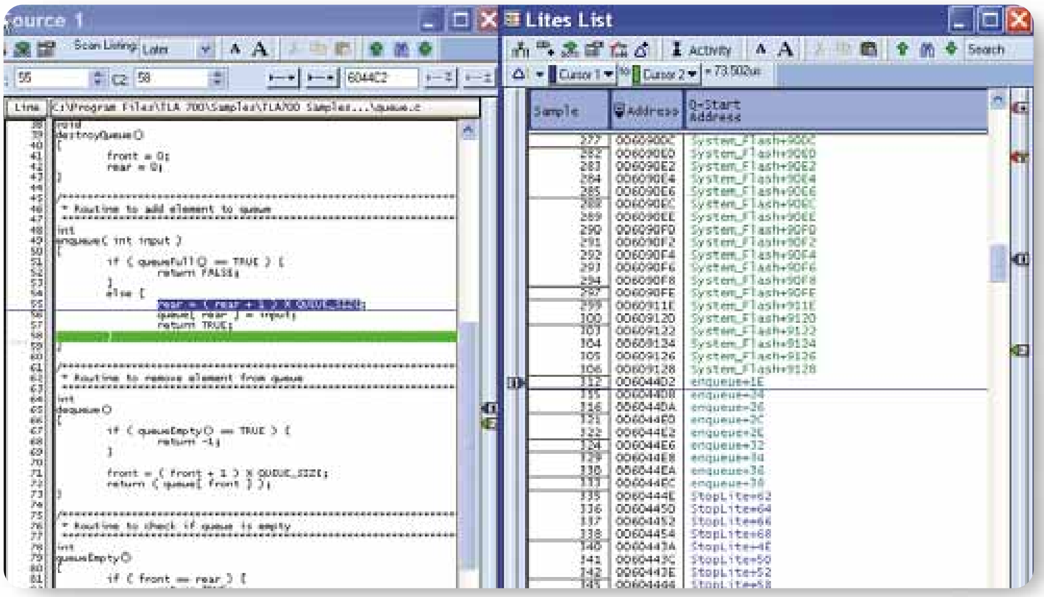
There are many factors that can impact timing performance in a digital circuit, necessitating an acute attention to detail when creating and producing today's designs. Two of the most common causes of timing errors can be found in the design and construction of digital circuits.

Design Causes

Due to the complexity of today's digital systems, human error is a regular occurrence during the design phase of system development. Misunderstanding the operation of purchased components, errors at the register transfer level (RTL) of intellectual property that have been implemented in FPGAs and incorrect hardware/software interaction are common design mistakes that can produce timing errors.



▶ **Figure 6.** *A simple circuit with clock skew and sample waveforms.*



```
Line  C:\Program Files\TLA 700\Samples\TLA700 Samples...\queue.c
void
destroyQueue()
{
        front = 0;
        rear = 0;
}

/*****************************************************************
 * Routine to add element to queue
 *****************************************************************
int
enqueue( int input )
{
        if ( queueFull() == TRUE ) {
                return FALSE;
        }
        else {
                rear = ( rear + 1 ) % QUEUE_SIZE;
                queue[ rear ] = input;
                return TRUE;
        }
}

/*****************************************************************
 * Routine to remove element from queue
 *****************************************************************
int
dequeue()
{
        if ( queueEmpty() == TRUE ) {
                return -1;
        }

        front = ( front + 1 ) % QUEUE_SIZE;
        return ( queue[ front ] );
}

/*****************************************************************
 * Routine to check if queue is empty
 *****************************************************************
int
queueEmpty()
{
        if ( front == rear ) {
```

```
Sample  Address   0-Start
                  Address
   277  0060900C  System_Flash+90DC
   282  006090E0  System_Flash+90E0
   283  006090E2  System_Flash+90E2
   284  006090E4  System_Flash+90E4
   285  006090E6  System_Flash+90E6
   288  006090EC  System_Flash+90EC
   289  006090EE  System_Flash+90EE
   290  006090F0  System_Flash+90F0
   291  006090F2  System_Flash+90F2
   292  006090F4  System_Flash+90F4
   293  006090F6  System_Flash+90F6
   294  006090F8  System_Flash+90F8
   297  006090FE  System_Flash+90FE
   299  0060911E  System_Flash+911E
   300  00609120  System_Flash+9120
   303  00609122  System_Flash+9122
   304  00609124  System_Flash+9124
   305  00609126  System_Flash+9126
   306  00609128  System_Flash+9128
   312  006044D2  enqueue+1E
   315  006044D8  enqueue+24
   316  006044DA  enqueue+26
   321  006044E0  enqueue+2C
   322  006044E2  enqueue+2E
   324  006044E6  enqueue+32
   329  006044E8  enqueue+34
   330  006044EA  enqueue+36
   333  006044EC  enqueue+38
   335  0060444E  StopLite+62
   336  00604450  StopLite+64
   337  00604452  StopLite+66
   338  00604454  StopLite+68
   340  0060443A  StopLite+4E
   341  0060443C  StopLite+50
   342  0060443E  StopLite+52
   345  00604446  StopLite+58
```

▶ **Figure 7.** *A logic analyzer's software debug window correlates source code with real-time hardware operations.*

One example is an improper clock distribution design resulting in setup/hold violations (see Figure 6). In this scenario, a sequentially adjacent register is used to pass a data input through two flip-flops on the same clock edge in a FPGA design. Due to limited global resource restrictions, the designer chooses not to use the clock network in the FPGA chip, instead connecting two flip-flop clocks to the clock signal utilizing a normal routing resource. This causes a delay of the clock signal arriving between the two flip-flops, which is called clock skew. When clock skew is not considered carefully according to the propagation delay on the data path, the second flip-flop may latch the D1 input logic before the appropriate clock edge arrives. Consequently, the D1 input passes through two registers faster than expected and causes timing errors. Improper clock skew may also cause setup/hold violations when the D2 switching edge is too close to the clock edge.

Many other factors, including system software, can also affect timing performance during the design phase. A system's software code must be executed precisely by the microprocessors and related digital systems; however, few software engineers are well versed in hardware operations. As such, most software is designed based on assumptions of what will happen once the code is executed over a circuit. Oftentimes the software may require more cycles for the hardware to respond than expected. This dynamic can create sequencing errors when the output is correlated with other subsystems.

Fortunately, logic analyzers are able to bridge the gap between the hardware and software realms. By correlating software code with real-time hardware operations, line-by-line, the logic analyzer delivers a comprehensive view and understanding of system operations (see Figure 7).

Structural Causes

After the design phase, engineers must test and debug their design on one or more prototype boards. Once fully tested, these boards are turned into models that are used by manufacturing teams to mass-produce them for market consumption. However, problems can arise during the manufacturing phase that were not evident during the initial testing of prototype boards. These constructive or structural variances, such as trace length variances or differences between boards and components used in the manufacturing process, can change prorogation delay values and cause timing errors. Therefore, engineers must implement proper design margins to tolerate these manufacturing variances.

Confronting Timing Errors

Although timing errors are common in today's digital designs, there are several tips and tools for efficiently resolving them, or avoiding them altogether.

Preempting Problems

Before starting a design, engineering teams need to meticulously consider their desired system, the resources available, the components being utilized and the course of development. In doing so, they must anticipate potential problem areas and make sure to address and prevent them before they become genuine problems.

Simply put, it is much better to confront problems earlier in the design and development cycle than later. Detecting and resolving functional and signal integrity issues as early as possible can greatly reduce debug time and development costs.

Two potential problem areas should be thoroughly considered before embarking upon system design: functional issues and signal integrity issues.

Functional Issues: Although common, functional problems can be caught and avoided through an effective and well-defined development process. For example, paper design reviews help examine power distribution and clock distribution to avoid timing errors. Board-level simulations and static timing analysis tools are also useful for identifying errors in the early stages of system design.

Signal Integrity Issues: Another common cause of timing errors is signal integrity issues produced by noise, distortion and other anomalies. Low-amplitude signals, slow transition times, glitches, overshoot, crosstalk, signal path design, impedances, loading, transmission line effects and even power distribution on the circuit board can impair a signal in the analog domain, which can then cause timing errors in the digital domain.

There are many advanced simulation tools that help reduce errors in the early stages of digital system design. However, these tools have inherent limitations, notably the simulation of flip-flop scenarios. For example, simulating the synchronization of signals across clock boundaries may not reveal problems such as fast path, race condition and hold violations. Because of these limitations related to simulation tools, it is advised that engineers create their designs in preparation for and support of debug and verification.

Designing for Debug and Verification

Much to the dismay of engineers who work tirelessly to create a flawless system, no design is perfect. Today's complex designs involve arduous debug work during the validation phase of system development. Engineers must be able to identify problems, trace their root causes and resolve them – quickly and efficiently. Experienced engineers will utilize the design phase to prepare their systems for the debug and verification phase, easing the process of detecting and eliminating problems.

This preparation can be accomplished by developing a test and validation plan in conjunction with circuit design to address the potential problems and issues. A robust test and validation plan helps remove surprises and potential roadblocks by:

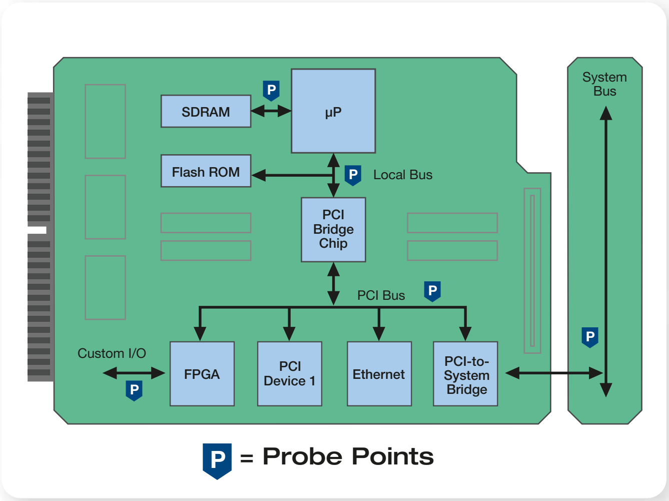
- ▶ Identifying functionality to be tested and how it will be tested
- ▶ Identifying interfaces and signals that need to be verified
- ▶ Identifying the type of measurements that need to be made

In the design phase, experienced engineers address several fundamental questions, such as:

- ▶ What are the potential issues when validating the timing sequence and specifications?
- ▶ What tools and banner specifications will be required to address those potential issues?
- ▶ How will system activities be correlated and sub-systems isolated so the debug scope can be narrowed and problem areas identified?

To answer these questions, engineers will need to begin by designing their circuits with an effective probing strategy, ensuring they will have total system visibility during the debug and validation phases of system development. In tandem with a successful probing strategy, powerful instruments can help engineers visualize system activities, correlate timing relationships among multiple interfaces, channels and waveform edges and provide accurate measurement information. With the right probing approach and high-quality instruments, engineers can identify problems and debug their circuits much more effectively (see Figure 8).

Component selection can also play an important role when developing an effective validation and debug plan. For example, selecting FPGA chips that support FPGAView can save time in validation and debugging. FPGA chips are widely used in many embedded system designs. Engineers often need to conduct timing analysis on the signal running inside the FPGA chip and correlate the internal signals with signal activities outside the chip. Therefore, using FPGA components that support FPGAView enables the engineer to conduct real-time debugging on FPGA chips more efficiently. FPGAView allows designers to view the internal operation of their FPGA and correlate the internal signals with other board signals. It also lets the engineer change internal probe points without recompiling their design to monitor multiple internal signals per debug pin.



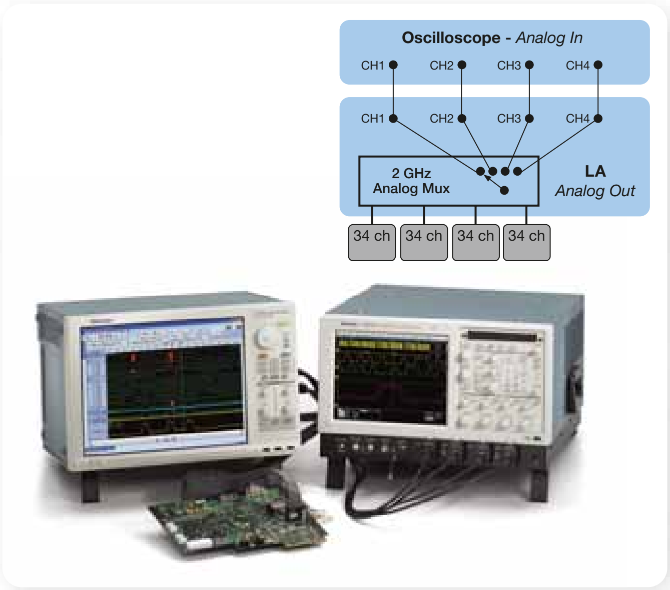
▶ **Figure 8.** *A probing strategy to enable total system visibility.*

Remember, an effective debugging plan starts in the design phase. A well-prepared debugging plan that is developed during the design cycle will reduce surprises and obstacles and expedite timing analysis work.

Finding Problems Efficiently

Although most engineers are very good problem solvers, they need clues that help them trace and understand system problems. Without clues, engineers' debugging efforts can be directionless and unproductive. This underscores the importance of test and measurement tools, namely oscilloscopes and logic analyzers, which enable engineers to visualize system activities and find clues regarding elusive timing errors.

Oscilloscopes are great tools for visualizing analog details, measuring characteristics such as rise time, fall time and jitter and examining system power, clocks and key signals. However, these powerful instruments have limited channel counts and logic trigger options. When conducting timing analysis on multiple channels or defining intelligent logic triggers, the logic analyzer is the preferred instrument. A logic analyzer is used to examine timing execution and signal integrity on multiple channels, buses and microprocessors.



▶ **Figure 9.** *iCapture™ multiplexing simplifies probing.*

To help confront the complexity of modern digital systems, recent innovations have boosted the debugging power of these traditional test and measurement instruments. The following innovations have proven effective in reducing the time and difficulty of debugging today's systems.

The iLink™ Toolset Advantage

Modern designs require the analysis of both analog and digital waveforms to validate system timing. Engineers must use logic analyzers to verify signal logic status, bus values, signal sequencing, setup/hold times and so on. In addition, they need an oscilloscope to make analog measurements such as rise time, fall time, jitter and pulse distortion.

To take advantage of the inherent strengths of each instrument, engineers can now integrate a logic analyzer and oscilloscope with an iLink toolset. This analog and digital integration solution speeds up debugging and verification work by wading through the digital information stream to trigger on circuit faults and capture related events with a logic analyzer. Concurrently, an oscilloscope can peer behind the digital timing diagrams to show the raw analog waveforms and measurements.

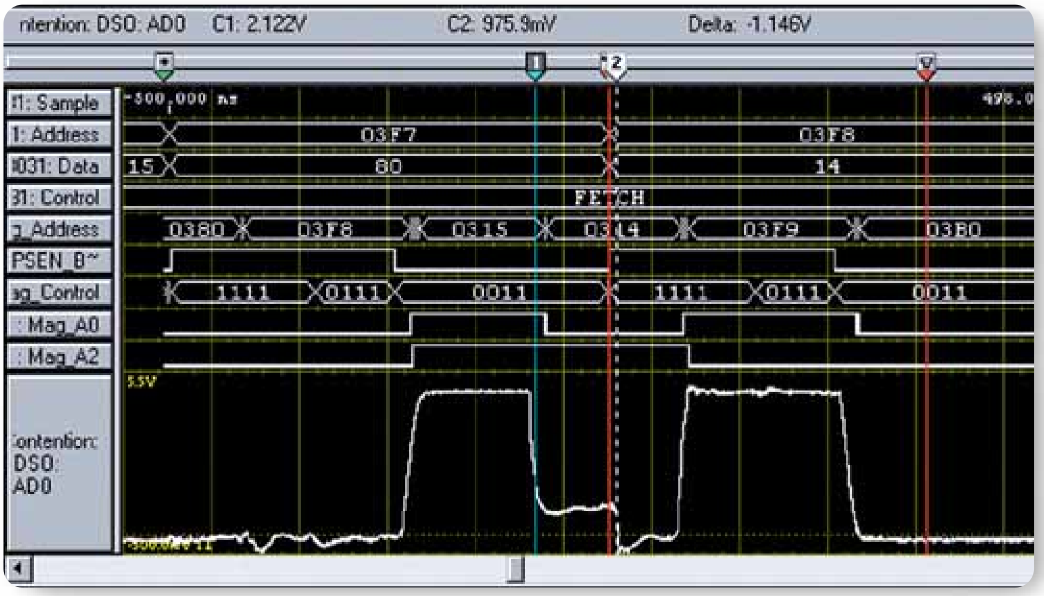
The iLink toolset, which includes iCapture™ multiplexing, iView™ display and iVerify™ analysis, is a comprehensive package that brings exceptional debugging power to engineers and speeds up waveform analysis work.

iCapture™ multiplexing provides simultaneous digital and analog acquisition through a single logic analyzer probe (see Figure 9). When examining signal trace waveforms, engineers can route any four channels among hundreds to an oscilloscope with a few mouse clicks. Since the analog outputs are always "live," engineers can save tremendous amounts of time and effort by not having to set up analog probing, channel-by-channel.

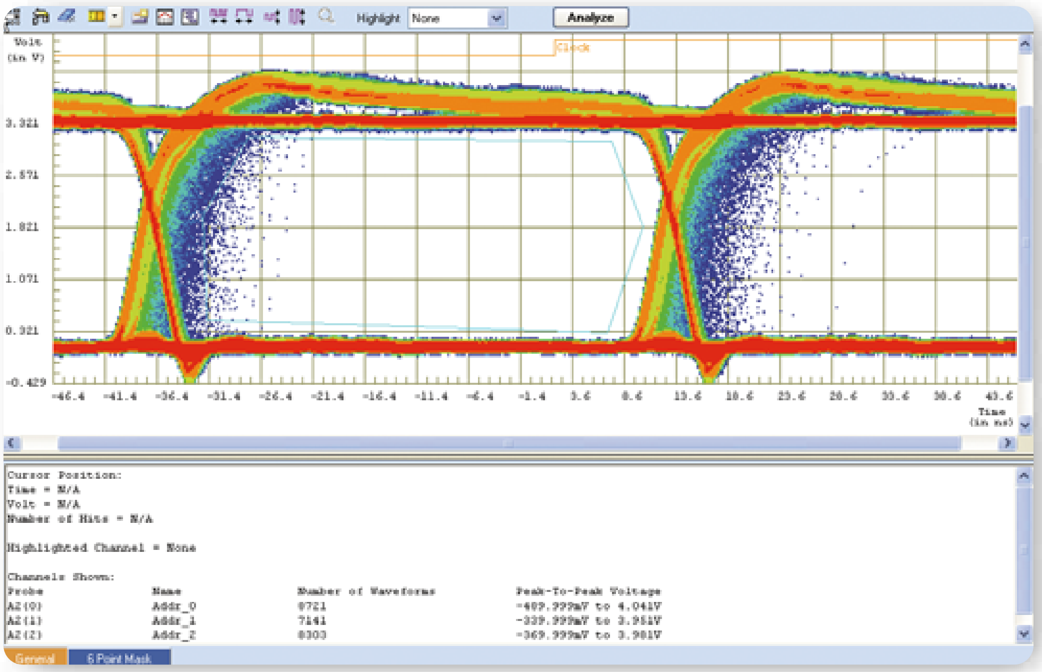
iCapture multiplexing also eliminates the double probing dilemma. Engineers no longer have to place both a logic analyzer probe and oscilloscope probe on a circuit at the same time. In some applications, the excessive loading from two probes can affect signal authenticity and impact measurement accuracy.

iView™ display delivers time-correlated, integrated logic analyzer and oscilloscope measurements on the logic analyzer display (see Figure 10). During digital timing analysis, engineers can use the logic analyzer's intelligent trigger and wide channel counts to capture elusive timing errors, while at the same time using the oscilloscope to view analog details of target signals. The iView display gives engineers the capability to cross trigger between the logic analyzer and oscilloscope and place the digital and analog waveforms on one screen in time-correlated fashion. For example, engineers can capture timing glitches from multiple channels with a logic analyzer. A red flag on the logic analyzer screen will specify which channel has a glitch. MagniVu™ high-resolution timing will unveil the glitch details. And time-correlated analog waveforms from the oscilloscope will reveal the cause of the timing glitch.

iVerify™ analysis provides multi-channel bus analysis and validation testing using oscilloscope-generated eye diagrams (see Figure 11). With iVerify analysis, engineers can quickly validate the parameters of a system through the oscilloscope's powerful acquisition and analysis capabilities.



▶ **Figure 10.** *iView™ display enables analog and digital waveforms on one screen.*



```
Cursor Position:
Time = N/A
Volt = N/A
Number of Hits = N/A

Highlighted Channel = None

Channels Shown:
Probe              Name               Number of Waveforms          Peak-To-Peak Voltage
A2(0)              Addr_0             8721                         -409.999mV to 4.041V
A2(1)              Addr_1             7141                         -339.999mV to 3.951V
A2(2)              Addr_2             8303                         -369.999mV to 3.981V
```

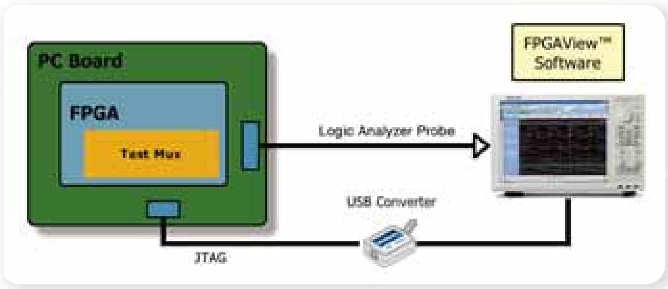
▶ **Figure 11.** *iVerify™ analysis provides multi-channel eye diagrams for easier bus analysis.*

The iVerify analysis capability automatically captures and displays a high-resolution eye diagram on the logic analyzer screen. Engineers can view eye diagrams across the entire bus, enabling them to easily verify their design in relation to data valid window specifications required by industry standards such as RapidIO and PCI Express. iVerify analysis also helps identify elusive signal integrity problems in the timing domain, such as crosstalk, skew, overshoot, slew rate problems and inter-symbol interference.

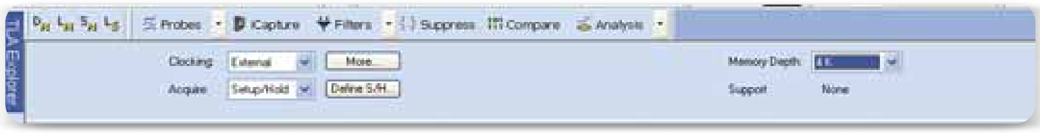
The FPGAView™ Software Advantage

FPGAs are increasingly popular in today's digital system designs. To validate system performance, engineers often need to use a logic analyzer to view internal signal activities and correlate them with other signals on the board. To do so, they must route internal channels to dedicated debug pins and view them with the logic analyzer.

This approach can be tedious and timing consuming. Pins on FPGA chips are typically scarce and debug pins are therefore very limited. If engineers wish to view different signals, they are forced to make changes in their RTL or use FPGA edit tools to reroute the desired set of signals to debug pins over and over again. This process is not only time consuming, but can also change system timing if it requires a recompilation of the design, potentially hiding the problem the engineer is attempting to resolve.



▶ **Figure 12.** *Typical FPGAView implementation.*



Clocking: External
Acquire: Setup/Hold
Memory Depth: [illegible]
Support: None

▶ **Figure 13.** *Changing the clocking mode.*

First Silicon Solutions has developed FPGAView software to operate in conjunction with Tektronix logic analyzers to eliminate this problem, helping engineers focus on signal analysis (see Figure 12). By using FPGAView, engineers no longer need to recompile their design every time they want to change the signals they are monitoring. Not only does this save compiling time, it also avoids changing system timing through different routing and placement.

FPGAView also preserves on-chip resources such as memory. Timing errors often only appear after a series of events, necessitating a long record length to capture. With FPGAView, engineers are able to take advantage of the logic analyzer's resources to save limited on-chip resources and control system complexity. Engineers can also use the logic analyzer to correlate internal signals with other signals on the circuit with FPGAView. By using the intelligent trigger resources and measurement features of a logic analyzer, timing errors can be captured quickly and easily. In addition, FPGAView automatically transfers the signal names from a design to the logic analyzer. This saves a substantial amount of time in retyping hundreds of channel names and matching them with the original design. It also increases the flexibility for monitoring system signals during timing analysis.

Application Examples

To highlight the concepts covered thus far, the following application examples provide a guide for debugging a few common timing problems.

Capturing Setup/Hold Violations

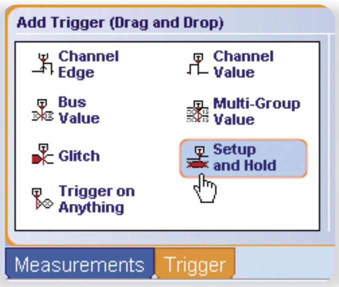
Setup/hold compliance is one of the most crucial timing parameters in a digital system. The traditional approach of verifying setup/hold timing using two oscilloscope channels and probing the clock and data lines one-by-one becomes tedious and time consuming when there are numerous signal traces running on the circuit. However, a logic analyzer can scan entire system buses to trigger on and display setup/hold violations automatically.

To simplify the process, engineers should create test points during the design phase that will allow the logic analyzer's probe to easily access the clock and target signals. Engineers should also consider logic analyzers that are able to clock system signals in timing mode and state mode simultaneously. State mode enables a logic analyzer to trigger on setup/hold violations, while a high-resolution timing mode enables the engineer to measure the violation. The Tektronix TLA Series logic analyzer is used in the following example.

Step I

First, the engineer must connect a probe to the target device and assign the appropriate logic threshold. The engineer must then go to the Setup Window to change the clocking mode to "External" (see Figure 13).

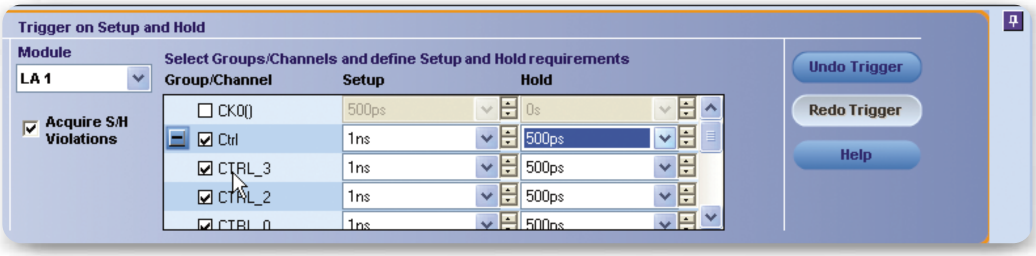
Performance logic analyzers have two ways to clock target signals: internal mode and external mode. In the internal mode, also called timing mode, the logic analyzer samples the signals of the device under test (DUT) periodically by using an internal clock. The user can adjust the sample rate to change the timing resolution.



▶ **Figure 14.** *Selecting the "Setup and Hold" trigger option.*

In the external mode, also called state mode, the logic analyzer samples acquired signals according to an external clock, which is typically synchronized with device signals. In this mode, logic analyzer users can see waveform states only when they are valid.

To capture setup/hold violations, the engineer must compare the waveform edge timing relationship between clock and target signals. Therefore, they need to use external mode to capture setup/hold violations.



Module: LA 1

☑ Acquire S/H Violations

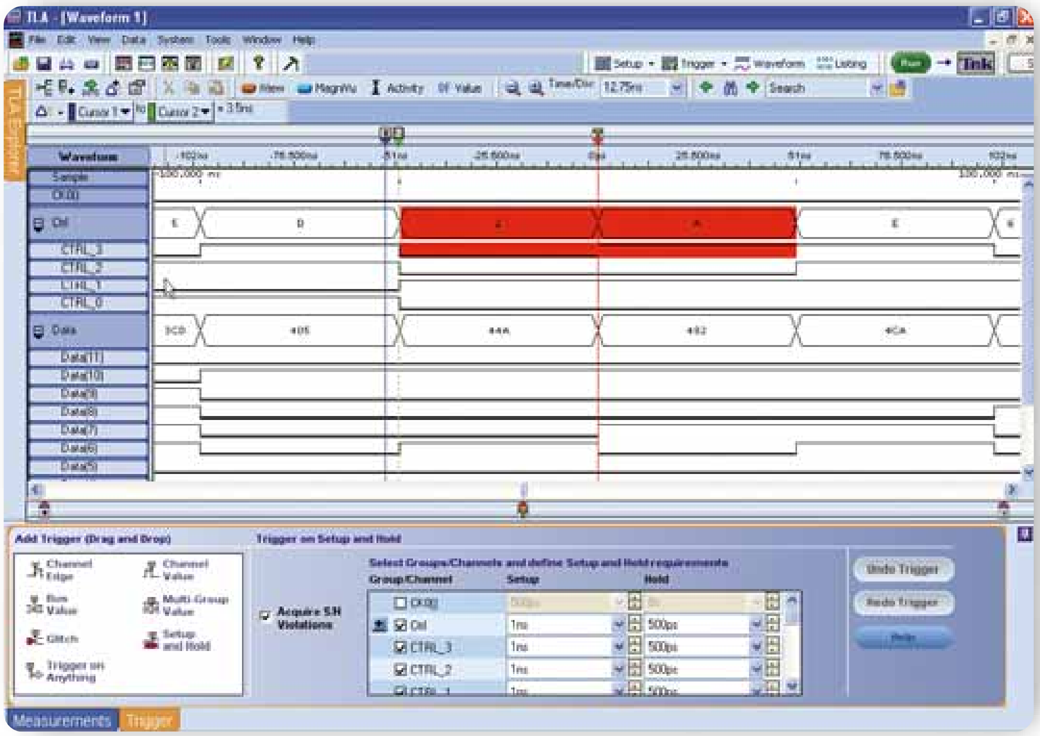
Select Groups/Channels and define Setup and Hold requirements

Group/Channel	Setup	Hold
☐ CK0()	500ps	0s
☑ Ctrl	1ns	500ps
☑ CTRL_3	1ns	500ps
☑ CTRL_2	1ns	500ps
☑ CTRL_0	1ns	500ps

▶ **Figure 15.** *Specifying setup and hold times.*

Step II

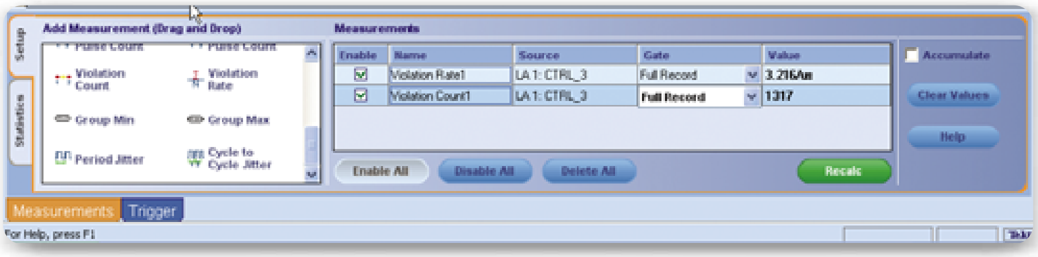
The next step is to drag and drop the "Setup and Hold" trigger option onto the target buses and define bus setup and hold specifications (see Figures 14 and 15). In this example, the data bus has a specification of 1ns setup time and 500ps hold time.



▶ **Figure 16.** *A red flag indicates a violation.*

Step III

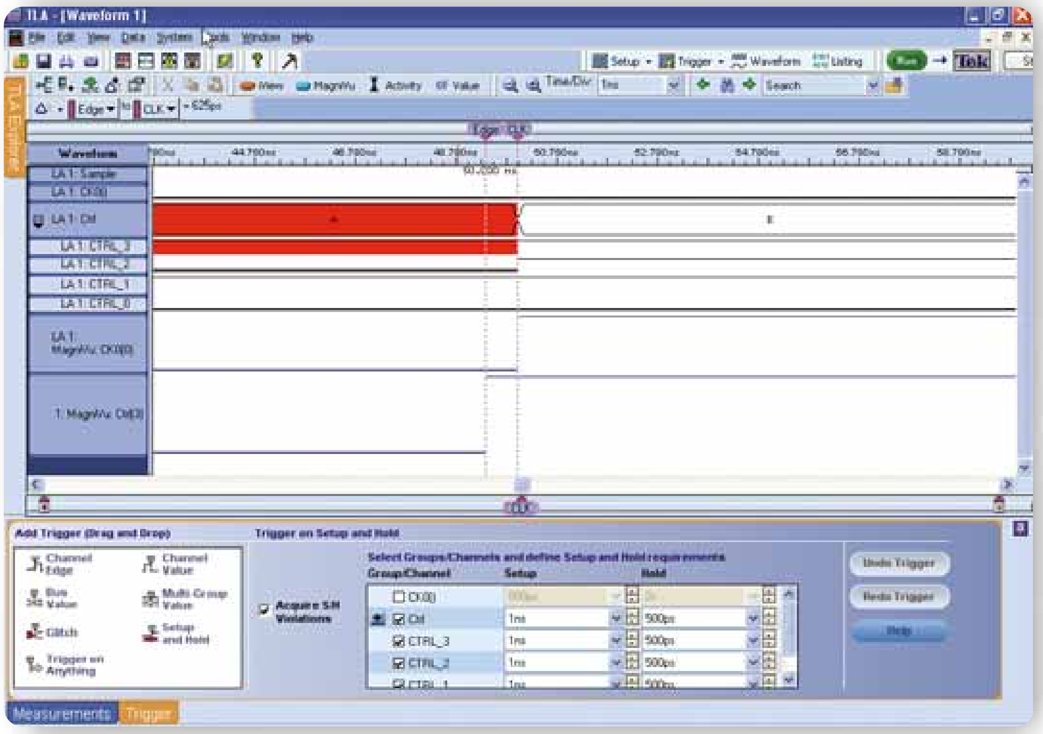
After defining the trigger condition, the engineer simply presses the "Run" button to begin a new acquisition. The logic analyzer automatically examines thousands of active waveform edges in accordance with the clock edge. As soon as the logic analyzer identifies a setup/hold violation, it will trigger and place red flags on the screen to display the violation areas (see Figure 16).



Measurements

Enable	Name	Source	Gate	Value
✓	Violation Rate1	LA 1: CTRL_3	Full Record	3.216A/s
✓	Violation Count1	LA 1: CTRL_3	Full Record	1317

▶ **Figure 17.** *Automated measurements identify the violation counts and rates.*



Trigger on Setup and Hold

Select Groups/Channels and define Setup and Hold requirements

Group/Channel	Setup	Hold
☐ CK0(0)	[illegible]	[illegible]
☑ Ctrl	1ns	500ps
☑ CTRL_3	1ns	500ps
☑ CTRL_2	1ns	500ps
☑ CTRL_1	1ns	500ps

▶ **Figure 18.** *MagniVu provides measurements and clues about a violation.*

This process enables the engineer to identify problem areas quickly. Plus, they can also use the logic analyzer's automated measurements to determine how many violations are in the system's buses (see Figure 17).

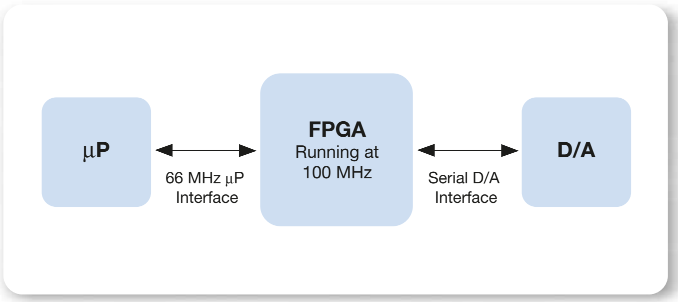
Step IV

Once setup/hold violations have been identified, the next step is to gather more details and make necessary measurements. Logic analyzers with high-resolution MagniVu acquisition are extremely helpful in this regard, along with the ability to acquire signals in timing mode and state mode at the same time from the same probe acquisition. MagniVu waveforms can be used to measure the delta time between the clock edge and signal edges to attain additional clues about the problem (see Figure 18). These clues provide a better understanding of the violation's root causes and aids in problem resolution.

Using FPGAView to Validate a FPGA

Although FPGAs offer flexibility in embedded system designs, validating and debugging the design inside a FPGA chip is still a challenging and time-consuming process. Timing simulation tools are available for FPGA debugging, yet they come with several inherent limitations. They have difficulty simulating real-world circuitry, especially when dealing with timing issues and interfaces across different clock domains. To more efficiently validate FPGA timing, engineers can use FPGAView to route internal signals out through FPGA pins and visualize them in a high-performance logic analyzer.

In this example, an Altera FPGA running at 100 MHz is used while interfacing with a microprocessor controller and DAC (see Figure 19). The µP interface is asynchronous to the 100 MHz FPGA clock. The serial D/A interface operates at an effective clock rate of 1 MHz. There is a register inside the FPGA that is the value/status register for the D/A. To



▶ **Figure 19.** *FPGA example with µP interface and serial D/A interface.*

drive a different analog voltage, the µP writes a new value to the D/A value register. This will clear the D/A READY bit in the status register telling the µP that the D/A control state machine cannot accept a new value. The FPGA then generates the appropriate serial data stream to the D/A. Once it is complete, the READY status bit is set, indicating the µP can write a new value to the value register.

For this particular example, the system works well with a few notable exceptions during the validation process. The value the µP is writing to the D/A value register is not reflected on the D/A output, and the simulations did not reveal any violations. To find the root cause of the irregular problem, the engineer must correlate the signal activities on the µP interfaces, inside the FPGA and on the D/A interface. With careful planning during the design phase, the µP, FPGA and D/A interfaces can easily be captured by a logic analyzer.

Step I

The engineer must first connect the logic analyzer to the µP interface and D/A interface. The logic analyzer is then set to trigger on a write to the D/A value register.

Step II

If no problems are identified when the µP and D/A interfaces are captured in this manner, the engineer can modify the logic analyzer's trigger. The modification may direct the instrument to trigger only if a write to the value register occurs and no serial data stream is generated in an appropriate amount time after the write. The engineer may find the logic analyzer triggers and shows that while a write occurs to the value register, no corresponding serial data stream is produced. This circumstance may not have been revealed during the simulation process.

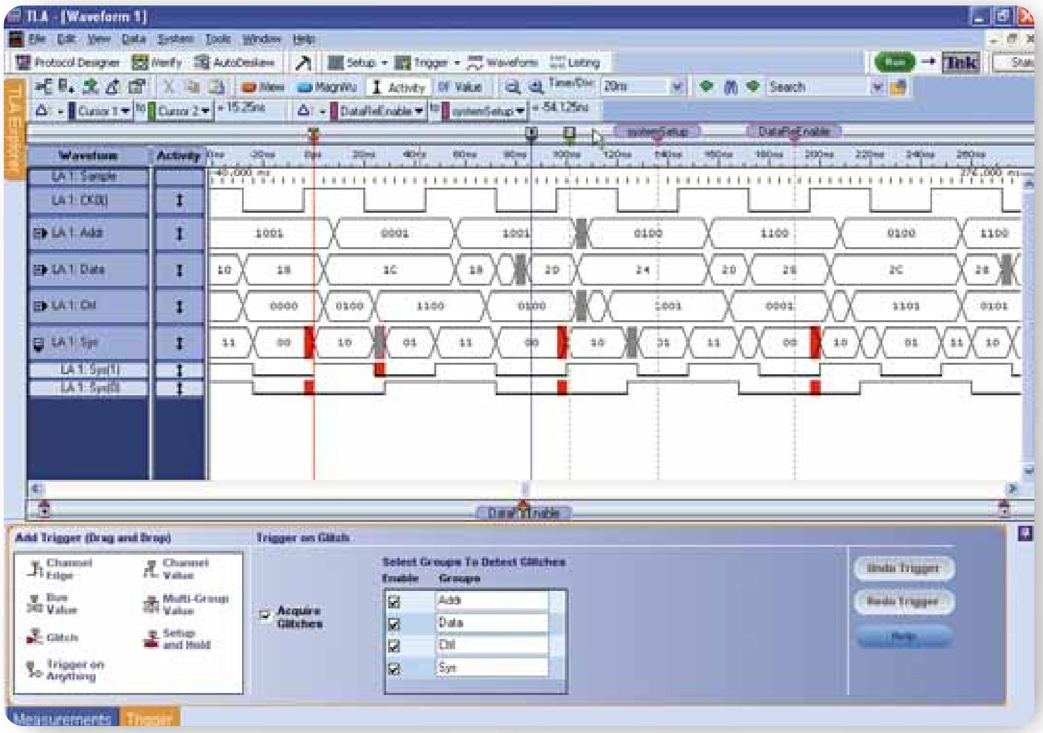
Step III

Once a problem has been identified, the engineer can use the FPGAView software to more easily analyze the internal operation of the FPGA. FPGAView provides the ability to select the D/A control state machine along with the µP and D/A interfaces, giving complete visibility of the design. The FPGAView software also programs the logic analyzer with FPGA internal signal names, assigning channels to make it easy to interpret measurement results. Correlating these FPGA signals with other signals in a system is done automatically by the logic analyzer.

By examining all of the data on the logic analyzer, the engineer will see the D/A control machine never starts, even though the system writes to the value register. When the new value is written to the value register, the state machine believes the READY bit is not asserted, indicating it was not ready for a new value. Because of this circumstance, the system ignored the write and did not even start the process of writing a new value to the D/A. At the same time, the READY bit to the µP was asserted, indicating it was okay to write a new value.

In this case, two different blocks of the design running at different frequencies were using the READY bit. Two different registers running at two different clock rates were sampling a single output. Since the D/A control machine runs much slower than the µP interface, the D/A control machine is not getting updated quickly enough. The engineer will need to change the system's RTL code to eliminate this problem.

This example highlights the timing challenges of today's digital designs, where signals are used in more than one clock domain. With careful planning during the design phase, engineers can gain the visibility needed to troubleshoot these potential timing problems. Seeing all of the key buses and FPGA internal signals enables engineering teams to effectively correlate system activities, visualize waveforms and isolate problems using logic analyzer triggers.



▶ **Figure 20.** *Red flags indicate problem areas on examined buses.*

Capturing Timing Errors Caused by a Glitch

Timing errors caused by glitches are usually characterized by intermittency and elusiveness. These characteristics make it difficult and time consuming to capture a glitch in a system with many channels. An efficient approach is to combine classic top/down troubleshooting with the specific advantages of the engineer's test instruments. To do so, the engineer must first take a broad, macro view of device operations and then begin focusing on particular problem areas.

On the macro (digital) level, the engineer can use a logic analyzer to perform glitch triggering on buses that contain hundreds of signals. The logic analyzer checks every signal for glitches, using red flags on the bus timing diagrams to highlight glitch locations. The engineer can then use an oscilloscope to help further characterize the problem by revealing exactly what the glitch looks like on a micro (analog) level. Using iView measurements on the Tektronix TLA Series logic analyzer, engineers can combine logic analyzer and oscilloscope displays into a single view to quickly solve the problem.

Step I

On the logic analyzer application waveform window, the engineer can drag and drop a glitch triggering option on the target buses. The logic analyzer's bus timing waveform is able to examine all of the bus signal lines at once. If the logic analyzer detects a glitch on any of the lines, it will place a red flag on the bus, the channel and the time location where the glitch occurs.

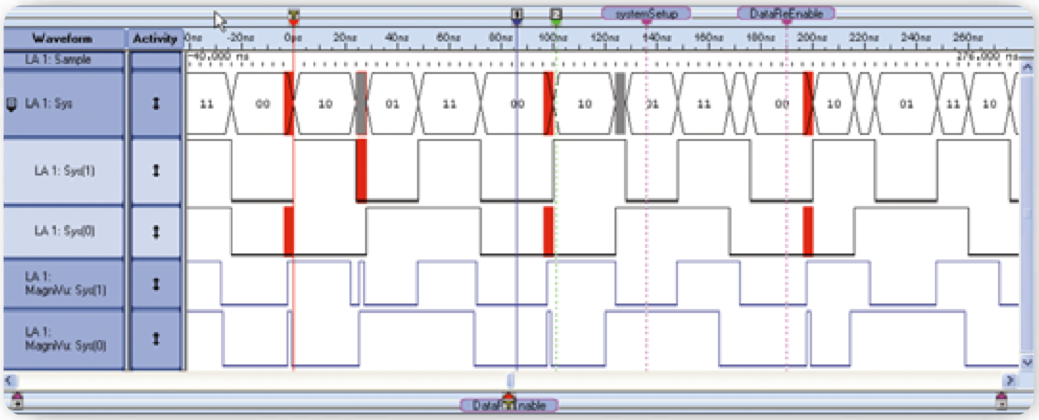
Step II

Once a glitch has been identified, the engineer can expand the logic analyzer's bus waveforms to view individual signals. The red glitch flags can be seen on signal lines Sys(0) and Sys(1) (see Figure 20).

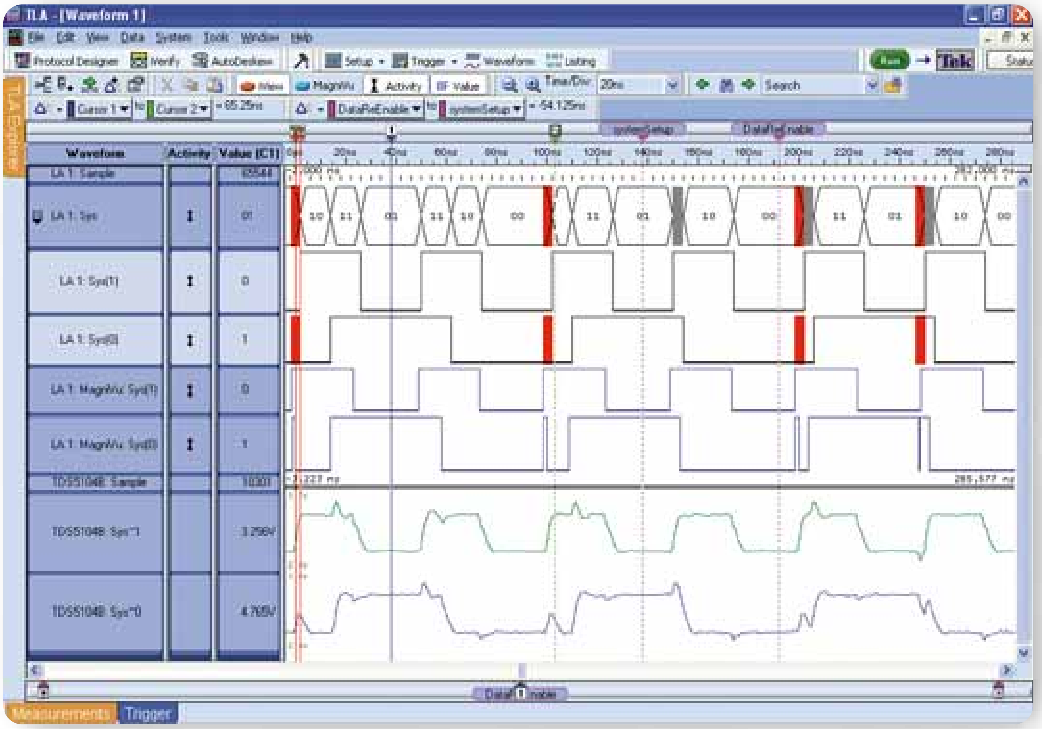
Step III

To see how the identified glitches relate to other events or faults, the engineer can use a high-resolution timing view to examine the faults in finer detail. Tektronix TLA Series logic analyzers offer high-resolution MagniVu timing acquisitions that run simultaneously with the instrument's deep timing. MagniVu waveforms can display all channels in high-resolution up to a 16 Kb memory depth. This is the equivalent of having two logic analyzers in one: a deep timing logic analyzer and a high-resolution timing logic analyzer, both using the same probes.

In this example, the engineer can use MagniVu high-resolution timing to examine the glitch identified in Sys(0) and



▶ **Figure 21.** *MagniVu acquisition unveils glitch details by using high-resolution timing waveforms.*



▶ **Figure 22.** *iView display correlates digital and analog domains.*

Sys(1). Since the logic analyzer's MagniVu waveforms are examining the signals at a much higher resolution (125 ps in this example), it is able to discern far narrower glitches on both lines (see Figure 21). Note that the glitch and a pulse occur at the same time on both signal lines. This often indicates crosstalk between the two signals, but the engineer will need to make a different type of examination to be sure.

Step IV

The engineer can discover what the glitch really looks like by comparing the analog and digital signal qualities using both an oscilloscope and logic analyzer with iView display capability. The iView display allows the logic analyzer to trigger the oscilloscope at exactly the right time to capture the glitch. With iView measurements, the logic analyzer also time-correlates the data and displays of both the analog and digital waveforms on the logic analyzer's display.

In this scenario, every leading edge of one signal has a corresponding positive voltage pulse on the other (see Figure 22). This makes crosstalk between Sys(1) and Sys(0) the obvious diagnosis. Crosstalk can easily occur on adjacent runs or pins within the system. High-frequency signals and clock edges have a greater susceptibility to crosstalk effects than lower frequency signals. This implies that consistently successful design practices at slower frequencies can be a contributor to failures at higher frequencies.

Logic analyzer glitch triggering can be used on buses with hundreds of signals. The instrument checks every signal line for glitches. If it flags a glitch, the engineer should start focusing on the problem to determine its source. This combination of cutting edge logic analyzers and oscilloscopes offers engineers a powerful tool for confronting timing errors caused by signal integrity problems.

Summary

High-speed buses, the popularity of FPGAs and the complexity of modern systems have rendered the process of timing analysis – a fundamental aspect of digital system validation and debugging – more complicated and time consuming than ever before. Today's systems are far more sensitive and vulnerable to timing errors, which are often elusive and hard to identify and resolve.

Fortunately, there are several tips and tools for efficiently resolving timing errors, or avoiding them altogether. By considering possible problem areas during the design phase, developing a thorough test and validation plan and employing an integrated oscilloscope and logic analyzer for a comprehensive view of system activities, engineers can effectively reduce the time and difficulty of timing analysis and system debugging.

For Further Information
Tektronix maintains a comprehensive, constantly expanding collection of application notes, technical briefs and other resources to help engineers working on the cutting edge of technology. Please visit **www.tektronix.com**

Tektronix

Enabling Innovation