Use capable triggering features to zero in on problems.

Doug Beck, Hewlett-Packard

Understanding Logic Analyzer Triggering

To debug an embedded system, you need to monitor the real-time behavior of its many signals and data streams, and logic analyzers still provide the most intimate look within an embedded system. However, logic analyzers don't work like magic; you must connect to the right signals and establish the right triggers to obtain the information you need for debugging.

Setting up logic analyzer triggers can be difficult and time consuming. You might assume that if you know how to program, you should be able to set up a logic analyzer trigger with no difficulty. However, this assumption could lead to a rude awakening as there are many concepts that are unique to logic analysis. Once these are understood, you can easily set up a logic analyzer to provide the needed data.

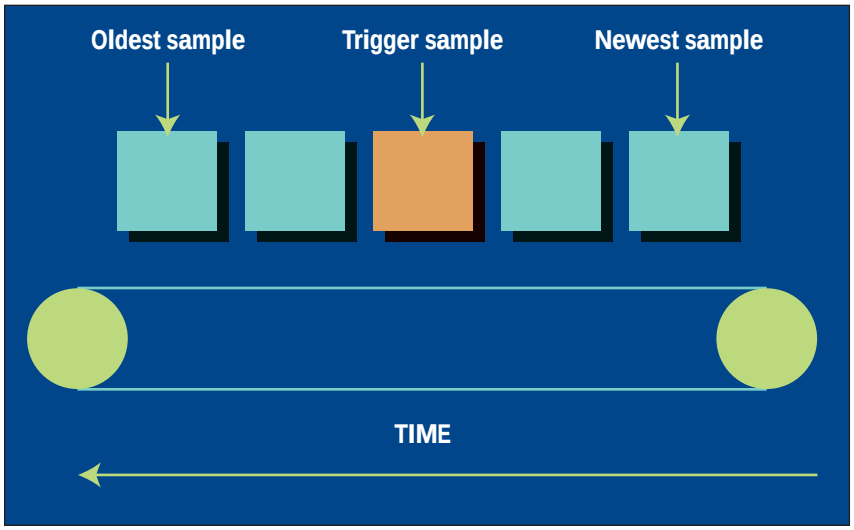

Figure 1. A logic analyzer is analogous to a conveyor belt. As new data enters, old data falls off the end.

The Conveyor Belt of Data

The memory of a logic analyzer can be compared to a very long conveyor belt, with samples acquired from the Device Under Test (DUT) as boxes on the conveyor belt (**Figure 1**). At one end new boxes are placed on the conveyor belt, and at the other end the boxes fall off. In other words, because logic analyzer memory is limited in depth (number of samples it can store), whenever a new sample is acquired, the oldest sample currently in memory is thrown away if the memory is full.

Continuing with the conveyor belt analogy, a logic analyzer trigger is similar to someone standing at the beginning of the conveyor belt who has been told to look for a special box and to stop the conveyor belt when that box reaches a particular position on the belt. The special box is the "trigger sample". Once a logic analyzer detects a sample that matches the trigger condition, it stops acquiring more samples when the trigger is located appropriately in memory. The location of the trigger in memory is the "trigger position". Normally, the trigger position is set to the middle so that an equal number of samples occurring before and after the trigger sample are in memory. However, the trigger position can be set to any point in memory.

Trigger Sequence

Table 1 is a summary of typical logic analyzer triggering capabilities. Although logic analyzer triggers are often simple, they can require complex programming. For example, you may want to trigger on the rising edge of one signal that is followed by the rising edge of another signal. Because there is a sequence of steps to find the trigger, this is known as a "trigger sequence". Each step of the sequence is called a "sequence level" or a "state".

Each sequence level consists of two parts, the conditions and the actions. The conditions are Boolean expressions such as "If ADDR = 1000" or "If there is a rising edge on SIG1". The actions are what the logic analyzer should do if the conditions are met. Exam-

CAPABILITY	EXAMPLES
Edges	If there is rising edge on SIG1 then Trigger If there is falling edge on SIG1 then Trigger
Boolean expressions	If ADDR = 1000 and DATA = 2000
Ranges	If ADDR in range 1000 to 2000
Storage qualification	1. If.. Else if ADDR in range 1000 to 2000 then Store Sample Go to 1 Else if ADDR not in range 1000 to 2000 then Don't Store Sample Go to 1
Counters	1. If DATA = 1000 then Increment Counter 1 Go to 2 2. If Counter 1 > 2 then Trigger
Timers	1. If DATA = 1000 then Start Timer 1 Go to 2 2. If Timer 1 > 500 ns then Trigger

Table 1. Summary of Logic Analyzer Triggering Capabilities.

ples of actions include triggering the logic analyzer, going to another sequence level, or starting a timer. This is similar to an If/Then statement in programming.

Each sequence level in the trigger sequence is assigned a number. The first sequence level to be executed is always Sequence Level 1, but because of Go To actions, the rest of the sequence levels can be executed in any order.

When a sequence level is executed for a sample and its conditions are not met, then the logic analyzer acquires the next sample and executes the same sequence level. Consider the following trigger sequence:

1. If DATA = 7000 Then Trigger

The logic analyzer will keep acquiring samples until the data has a value of 7000, and then it will trigger. Once a logic analyzer triggers, it will not trigger again, even if more than one sample meets the trigger condition.

If the conditions in a sequence level are not met, the logic analyzer will acquire the next sample and execute the same sequence level again. A trigger condition of "ADDR = 7000" is equivalent to "Keep acquiring more samples until you find one that has ADDR = 7000". If you set up a trigger condition that is never met, the logic analyzer will never trigger.

If a sample meets the condition, another sample is always acquired before the next sequence level is executed. Therefore it is not possible for a single sample to be used to meet the conditions of more than one sequence level, and each sequence level represents events that occur at different points in time. Two sequence levels can never be used to specify two events that happen simultaneously. For example, consider the following trigger sequence:

1. If ADDR = 1000 Then Go To 2
2. If DATA = 2000 Then Trigger

If the following samples were acquired, the logic analyzer would trigger on sample 7.

Sample No.	ADDR	DATA
1	1000	2000
This sample meets the condition in Sequence Level 1		
2	1010	3000
3	1020	4000
4	1030	5000
5	1040	6000
6	1050	7000
7	1060	2000
This is where the logic analyzer triggers		

The logic analyzer will not trigger on Sample 1 because a new sample is acquired between the time that the condition in Sequence Level 1 is met and when the condition in Sequence Level 2 is tested. A good way to think of this trigger sequence is "Find ADDR = 1000 followed by DATA = 2000 and then trigger".

Where to go next

When a sequence level's conditions are met and there is a "Go To" in the actions, it is clear which sequence level will be executed next, but if there is no "Go To", the next sequence level to be executed depends upon the logic analyzer's implementation. On some logic analyzers, if there is no "Go To" the next sequence level is executed. On others, the same sequence level is executed again. Because of this ambiguity, it is good practice to specify a "Go To" action rather than relying on the default.

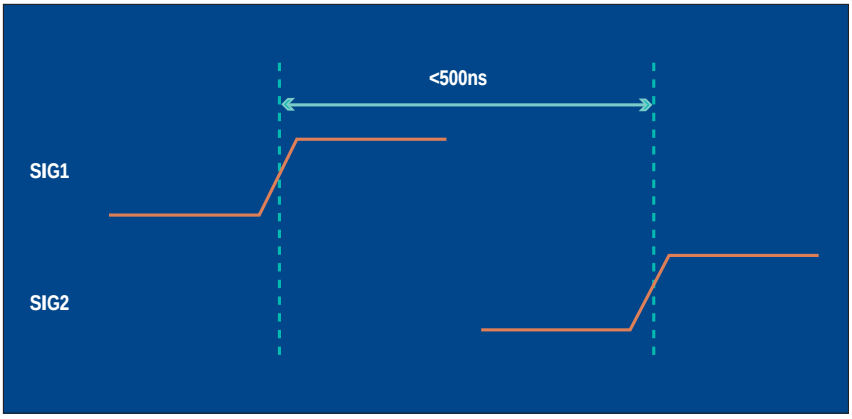

Figure 2. Locating some events requires the element of time. In this case, a trigger occurs when the edge of SIG2 follows the edge of SIG1 within 500 ns.

If you need to trigger on multiple conditions occurring simultaneously, you can use a Boolean expression within a sequence, such as "If ADDR = 1000 And DATA = 2000". For this condition to be met, ADDR must equal 1000 in the same sample that DATA equals 2000.

It is a common mistake to try to use two sequence levels when a Boolean expression should be used or to use a Boolean expression when two sequence levels should be used.

Boolean expressions are used for events that happen at the same time, and multiple sequence levels are used when one event follows another.

Branching

A branch is similar to the Switch statement in the C language and the Select Case statement in Basic. It provides a method for testing multiple conditions, each with its own actions. A Branch example is shown in the next column:

1. If ADDR < 1000 Then Go To 2
   This is a branch of Sequence Level 1
   Else If ADDR > 2000 Then Go To 3
   This is a second branch of Sequence Level 1
   Else If DATA = 2000 Then
   Trigger *This is a third branch of Sequence Level 1*
2. If DATA <= 7000 Then Trigger
3. If there is a rising edge on SIG1 Then Trigger

In Sequence Level 1 there are three branches, so there are three possible actions. When the condition of one branch is met, all of the branches below it are not tested: Each branch is an "Else If". Only one branch can be executed based upon a single sample, even if the sample causes the conditions for more than one branch to be met.

Another method to zero in on the data you need is to wait for an event to occur a specified number of times before triggering. Occurrence Counters are used when you want to find the nth occurrence of an event. For example, if you want to trigger on the fifth time that ADDR = 1000, you could set up the trigger as:

1. If ADDR = 1000 occurs 5 times Then Trigger

Using Time to Trigger

In some cases, you are interested in when something happens with respect to other events. Timers are used to check the amount of time that has elapsed between events. If you want to trigger on one edge followed by another edge that occurs within 500 ns (**Figure 2**), then a timer should be used. The most critical point to remember in using timers is that they must be started before they are tested.

The key to setting up a timer is to identify where it should be started and where it should be tested. Consider the example in Figure 2. The timer should be started when the rising edge on SIG1 is detected, and it should be tested when the rising edge occurs on SIG2. It may seem that the proper setup for this measurement is:

1. If there is a rising edge on SIG1, Then
   Start Timer1
   Go To 2
2. If there is a rising edge on SIG2 And Timer1 < 500 ns Then Trigger

The above trigger sequence actually has a critical flaw. What happens if there is a rising edge on SIG1 but SIG2 doesn't occur within 500 ns? The logic analyzer will never trigger, because Timer1 will keep running and the condition "Timer1 < 500 ns" will never be met. Later on, there might be another rising edge on SIG1 that

is followed within 500 ns by the rising edge on SIG2.

To fix this problem, whenever the timer exceeds 500 ns without triggering, the sequence should loop back to Level 1 to look for another rising edge on SIG1. The correct sequence is:

1. If there is a rising edge on SIG1, Then
   Start Timer1
   Go To 2
2. If there is a rising edge on SIG2 And Timer1 < 500 ns Then Trigger
   Else If Timer1 >= 500 ns Then
   Reset Timer1
   Go To 1

Take Only What You Need

To efficiently use the memory in the logic analyzer, you may want to store only those samples you are interested in. Storage qualification is used to specify whether an acquired sample should be placed in memory or discarded. The simplest method to establish storage qualification is to set up the Default Storage. This is specified separately from the trigger sequence, such as in a separate tab or another dialog. Default Storage establishes which samples are stored unless a sequence level specifies otherwise. For example, you may want to store samples only if ADDR is in the range 1000 to 2000, so the Default Storage should be set to "ADDR In Range 1000 to 2000".

Sequence-level storage qualification overrides the Default Storage and establishes which samples are stored within a particular sequence level. This storage qualification applies until a "Go To" or "Trigger" action is used to leave the sequence level. This is useful when you need a different storage qualification for each sequence level. For example, you may want to store nothing until ADDR = 1000 and then store only samples with ADDR in the range 1000 to 2000 for the rest of the measurement.

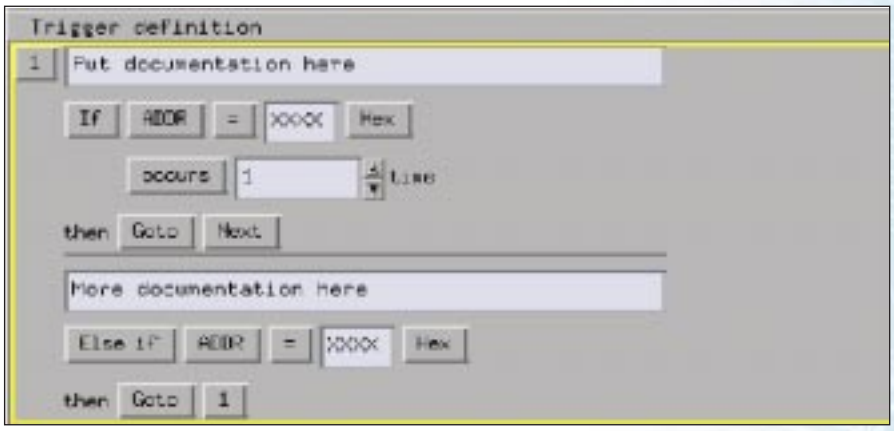

Figure 3. Documenting trigger sequences allows you to easily use them again.

Setting up sequence level storage requires the use of an additional branch. The following sequence level could be used to store only samples with ADDR in the range 5000 to 6FFF while looking for DATA = 005E:

1. If DATA = 005E Then Trigger
   Else If ADDR In Range
   5000 to 6FFF Then
   Store Sample
   Go To 1

Note the use of the Store Sample action. It means "store the most recently acquired sample in memory now", not "from now on, start storing". Because the Store Sample action is never executed unless ADDR is in the range 5000 to 6FFF, this branch essentially means "While in this sequence level, store only samples with ADDR between 5000 and 6FFF".

Simplifying the Setup

While setting up logic analyzer triggers can be difficult, trigger functions greatly simplify the process. Trigger functions are commonly needed building blocks that can be combined to set up a trigger sequence. You can set up a commonly used trigger sequence simply by selecting the appropriate function and filling in the data.

The most difficult part of setting up a complex trigger is often breaking down the problem. How do you map a complex trigger into sequence levels, branches, and Boolean expressions? Here are step-by-step instructions:

1. Break down the problem into events that don't happen simultaneously. These correspond to sequence levels.
2. Scan the list of available trigger functions to try to find some that match the events identified in Step 1.
3. Break down the remaining events into Boolean expressions and their corresponding actions. Each Boolean expression/action pair corresponds to a separate branch within a sequence level. Remember that "Store" branches can be used to handle storage qualification only for that sequence level.

Document your Trigger Sequences

If a trigger sequence is important at one time, it is likely to be important again, so documenting trigger sequences is a valuable investment in time. Complex trigger sequences generally are difficult to understand without some accompanying explanation, so inline documentation is important, as shown in **Figure 3**. Inline documentation is included in the trigger definition itself, allowing you to describe how different parts of the trigger work.

Setting up logic analyzer triggers is very different than writing software. The job can be greatly simplified if other work can be leveraged by using pre-defined trigger functions and previously written triggers that are well documented. Write your own trigger sequence only if nothing else is available. Finally, when faced with a difficult trigger to set up, break down the problem into smaller chunks and deal with each one separately.

Intuitive User Interface Improves Your Productivity

The capabilities of modern logic analyzers must expand along with the ever-increasing complexity of the systems being created. HP's new VisiTrigger technology allows you to easily use these added capabilities even if you don't use your logic analyzer every day. You can spend more time making measurements and less time paging through manuals and setting up the logic analyzer.

VisiTrigger technology is available in the HP 16600A/16700A logic analysis systems. It combines increased trigger functionality with a user interface that is easy to understand and use. With VisiTrigger, capturing complex events is as simple as point-and-click to choose the trigger function and fill-in-the-blank to customize it to your specific task.

Three new state-and-timing modules enhance the VisiTrigger technology with the addition of four-way branching and global counters. The HP 16715A, 16716A, and 16717A state-and-timing modules offer up to 333-MHz state and up to 2-GHz timing on each channel all of the time. See page 34 for more information on these new modules.

For additional information on products mentioned in this article, check 5 on the reply card, or visit http://www.hp.com/info/insight3.