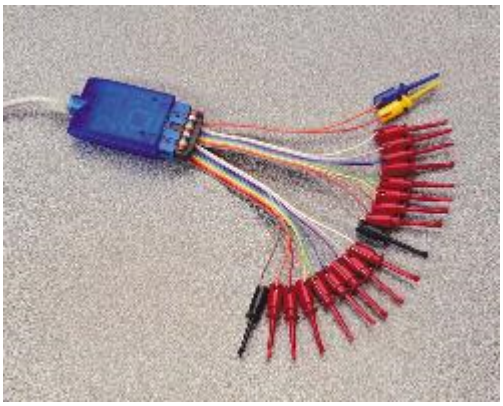
RockyLogic

Like the Ant8, the **Ant16** logic analyzer is compact, easy to use, powerful, and inexpensive. The Ant16 also features more channels (16), a synchronous acquisition mode, and Trigger-In and Trigger-Out connections.

The Ant16 has been superceded by the Ant18e and it is out of stock. It is possible that the Ant16 is in stock at our distributors EasySync. EasySync have offices in Portland, Oregon, USA and Glasgow, Scotland, and local representation in France, Germany, Spain and the Scandinavian countries.



Key Features

- Ultra-portable - fits in a pocket
- Powered from USB port - no additional power supply required
- 16 Channels, 2048 samples deep

- 500MHz asynchronous sampling speed
- 100MHz synchronous sampling speed
- Trigger In and Trigger Out connections
- Applications run on any recent version of Windows
- Inexpensive
- Simple or Complex triggering
- Software upgradeable
- View captured traces on the display of your PC

Control Panel Software

- Familiar Windows interface
- Data saved as plain text or in CSV format
- Step-by-step instrument setup and trigger setup.
- Upgradeable via the internet

Click here to download and evaluate the Ant control panel software.

Click here for screenshots from the control panel software.

PC Requirements

- USB socket. Either a USB socket on the PC or a socket on a *powered* USB Hub
- Windows 98, ME, 2000, or XP

More information

Click here for detailed specifications.

Please note that the Ant16 is high-performance test equipment. It is designed for users who are familiar with electronic design and debugging, and familiar with the usual safety precautions when working with electronic equipment.

RockyLogic

This is the specification of the Ant16 logic analyzer module. Additional technical information is presented in the FAQ section.

Acquisition Logic

Asynchronous Sample Rate	500MHz maximum, 100Hz minimum. 5-2.5-1 sequence.
Synchronous Sample Rate	100MHz maximum.
Channels	16, numbered 0 to 15
Memory Depth	2048 samples per channel
Threshold	0.8V to 2.5V, in steps of 0.1V
Minimum Input	0.5V below threshold for Lo, 0.5V above threshold for Hi
Trigger Connections	*Trigger In* and *Trigger Out*. *Trigger In* is only available in asynchronous clocking mode (using the internal clock). In synchronous clocking mode, this connection becomes *Clock In*.
Input Skew	less than 2ns, channel to channel
Input Impedance	100Kohms in parallel with 10pF
Maximum Input Voltages	+40V down to -40V

Triggering

Conditions	0, 1, Rising Edge, Falling Edge, Either Edge, and DON'T CARE for all channels.
Pattern Recognisers	2
Edges	Trigger on the condition becoming TRUE or on the condition becoming FALSE
Pass Count	0 to 1023
Trigger Logic	Multi-state trigger logic with Edge, Pattern, and Complex triggering. Complex triggering includes occurrence counting and minimum and maximum duration measurement.
Trigger Position	10% to 90% of acquisition memory, in 10% steps

Environmental

Operating Temperature	5°C to 40°C
Storage Temperature	-40°C to 75°C
Size	65 mm x 35mm x 15mm (2.5" x 1.4" x 0.6")
Power Consumption	2.2W maximum
Power Source	Via USB cable. No external power supply.
Computer Connection	USB connection to computer. Includes an attached USB cable.

Software Features

Host Computer	Logic Analyzer front panel software included for Windows 98/ME/2000/XP
Print Waveforms	Yes
Data Output	Text File and CSV File
Cursors	1 main cursor plus 1 auxiliary cursor

AN-2: Triggering

In this Application Note we will describe the triggering mechanism implemented in all RockyLogic logic analyzer products. The logic analyzer will be referred to as the Ant. This avoids having to write Ant8/Ant16 throughout the Note. All the remarks apply to the Ant8, allowing for the lower pin count and absence of ClockIn on the smaller product.

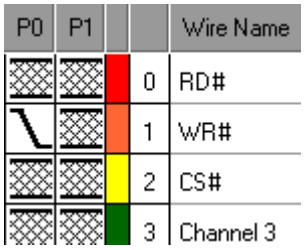
A trigger is an event you want to examine. Maybe you want to see what is happening on various wires when a write strobe goes active, in which case the event is (write strobe goes active.) Maybe you want to see what is happening when a chip is selected and the write strobe goes active, in which case the event is (chip select is active and write strobe goes active.)

You want to see what happens before the event, you want to see the event itself, and you want to see what happens after the event. So the job of a logic analyzer is to sample its incoming wires and continuously fill its memory buffer until the trigger event happens, then to continue sampling for a while so that post-trigger data is collected, then to stop.

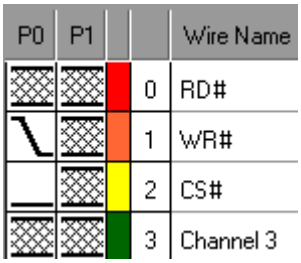
Although 99% of logic analyzer usage employs simple triggering, the Ant can also be fired by a complex triggering sequence. This note describes the possibilities.

Simple Triggering – Looking for the Write Strobe

A typical embedded microprocessor signals a write cycle by driving its WR# strobe low. We can record this event by triggering on the falling edge of the WR# strobe, as shown below. This is a pattern trigger, the pattern being *falling edge* on wire 1 and *don't care* on the other wires. P0 and P1 in the illustration refer to the two pattern recognisers – this example only uses P0.



WR# falling edge

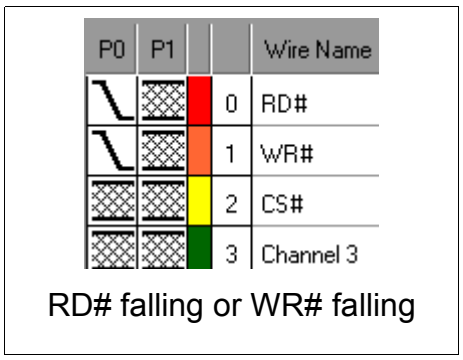


CS# low and WR# falling

The second illustration shows the setup when we want to trigger on WR# falling edge if and only if CS# is low. In other words, WR# *falling* and CS# *low* and other wires *don't care*.

Triggering on Read or Write

This sounds easy – set the trigger pattern as shown below. But we want to trigger on WR# falling or RD# falling. The conditions are OR'd together, whereas they were AND'd together in the previous example. The solution is that this is an *Edge Trigger* (which has its terms OR'd together), whilst the previous example was a *Pattern Trigger* (which had its terms AND'd together.) OR'd terms=edge triggering and AND'd terms=pattern triggering is broadly logical, but not the most memorable of terminology.

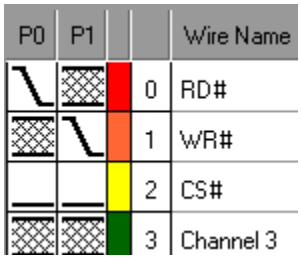


RD# falling or WR# falling

edge trigger for ORing

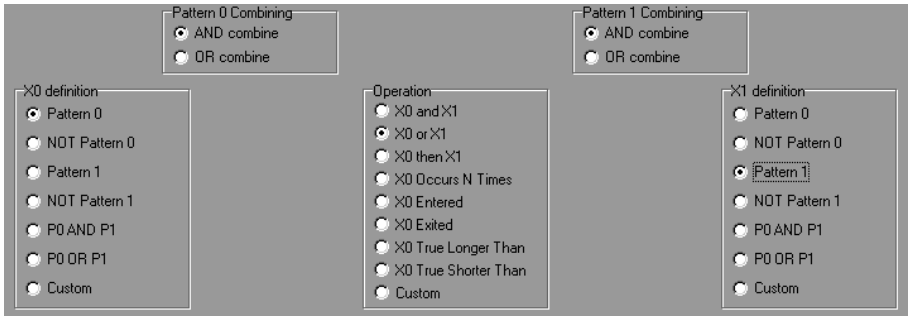
An Advanced Trigger – (Read or Write) plus Chip Select

Now we are interested in the first bus event, either a read or a write. One way to express this as a trigger condition is (CS# low and RD# falling) or (CS# low and WR# falling). Most logic analyzers have more than one pattern detector, so we set the first pattern detector to (CS# low and RD# falling) and the second pattern detector to (CS# low and WR# falling), as shown below:



read or write

Then we select and *advanced triggering* option to combine the two pattern detectors as a composite condition, as shown below:

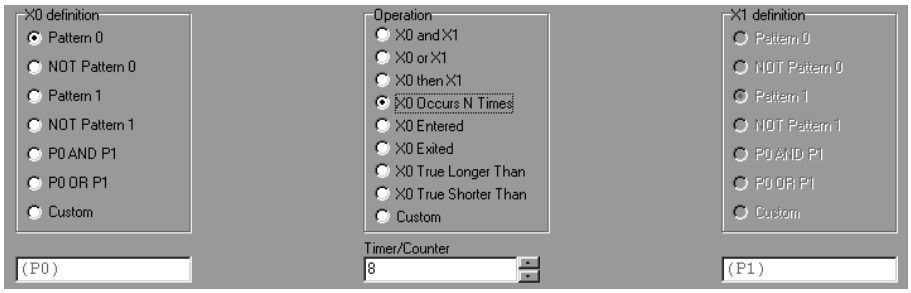


pattern 0 or pattern 1

We select AND combining for both patterns, then select X0=a hit on pattern 0 and X1=a hit on pattern 1. Finally selecting the trigger as (X0 or X1) gets us what we want. Of course, there are a number of other choices we could have made to end up with the same triggering condition.

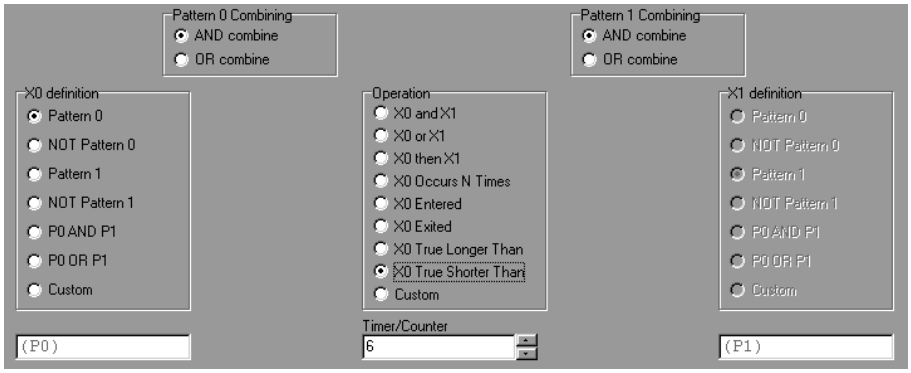
The Counter – Triggering on the 8th Write

This is pretty straightforward with an Advanced Triggering option. Here is the setup:



trigger when X0 occurs 8 times

Similarly, we can catch a minimum pulse width violation with this setup:



trigger when X0 narrower than 6 clock ticks

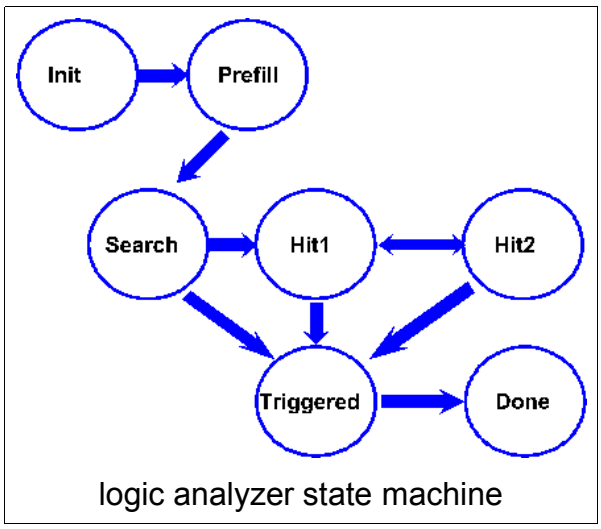
In this case we calculate the counter/timer value by dividing the clock speed. For instance, 60ns with a 100MHz (10ns) clock, gives a counter value of 6.

The Trigger State Machine

What is happening when we select the various triggering options?

First off, the pattern recognizer options are just setting various register bits which drive the combinational logic within the logic analyzer – this is how the logic analyzer knows it should be searching for particular patterns, ANDing or ORing the terms within the pattern recognizers, and combining the pattern recognizers to make overall trigger conditions.

But we are also setting bits which will drive the fundamental state machine which drives the data capture process. Here is a bubble representation of that state machine:



logic analyzer state machine

The transitions between the states are:

- Idle State: Move to Prefill when Run is signaled from the Host PC.
- Prefill State: Move to Search when the defined prefill percentage of the acquisition memory has been filled
- Search State: Move to Triggered if the *SearchTriggered* condition has been seen. Else move to Hit1 if the *SearchHit1* condition has been seen.
- Hit1 State: Move to Triggered if the *Hit1Triggered* condition has been seen. Else move to Hit2 if the *Hit1Hit2* condition has been seen.
- Hit2 State: Move to Triggered if the *Hit2Triggered* condition has been seen. Else move to Hit1 if the *Hit2Hit1* condition has been seen.
- Triggered State: Move to Done when the defined postfill percentage of the acquisition memory has been filled.
- Done State: Move to Idle when Reset is signaled from the Host PC.

So programming this state machine amounts to defining the conditions which cause it to change state. For instance, take the simple trigger with which we started this note – trigger when a particular pattern is seen. In this case we define X0 as a hit on Pattern Recognizer P0 and set SearchTriggered=X0.

A more complex example would be to trigger when X0, however defined, becomes false for the first time. We want to set the state machine to traverse this path: Idle->Prefill->Search->Hit1->Triggered->Done. So we set SearchHit1 = X0 and Hit1Triggered = (!X0).

Programming the Trigger State Machine

The RockyLogic software defines eight advanced triggering acquisition options. So how do we proceed if what we want is not one of the magic eight options? For instance, maybe we want to arm the logic analyzer with P0 and trigger when P1 goes from true to false. What we need is to set X0=P0, X1=P1 and

```
SearchHit1   = X0
Hit1Hi2      = X1
Hit2Trigger  = !X1
```

This is not a standard triggering option, but we can program it by inputting the equations directly via the Advanced Triggering window. In the RockyLogic implementation each condition is an equation of up to four variables. The variables are always called I0, I1, I2, and I3, and the meaning of these variables is described in the help files and via context sensitive help. For example, the *Hit2Triggered* equation is a function of I0=X0 and I1=X1, I2 is the timer counter, and I3 is the TrigIn input. Since we want to have Hit2Triggered=!X0 we set the

Hit2Triggered box to (~I0). Similarly we set the SearchHit1 box to (I0) and the Hit1Hit2 box to (I1).
And that is all. We have now programmed the logic analyzer for our non-standard acquisition.