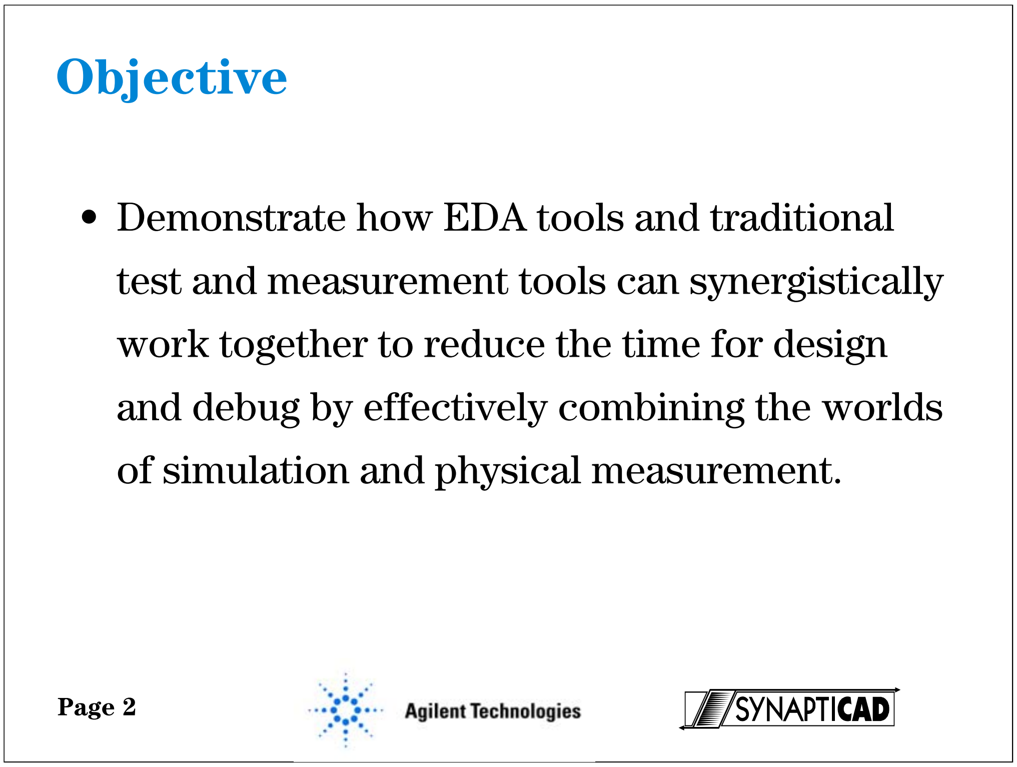
EDA-Assisted Hardware Verification

Agilent Technologies
Innovating the HP Way

SYNAPTICAD

Designers are looking for new tools that will reduce design time and eliminate multiple hardware prototypes. In many ways EDA tools appear to be the answer.

Unfortunately, many projects are delayed when simulation alone fails to create the perfect design. The problem is not in the EDA tools but in the unpredictability of the physical world. Simulation to the exclusion of physical measurements is as inefficient as the traditional design – debug – redesign loop. Today's design teams need tools that work together. This paper demonstrates how EDA tools and physical measurement tools can work together to reduce the design time of your product.



Objective

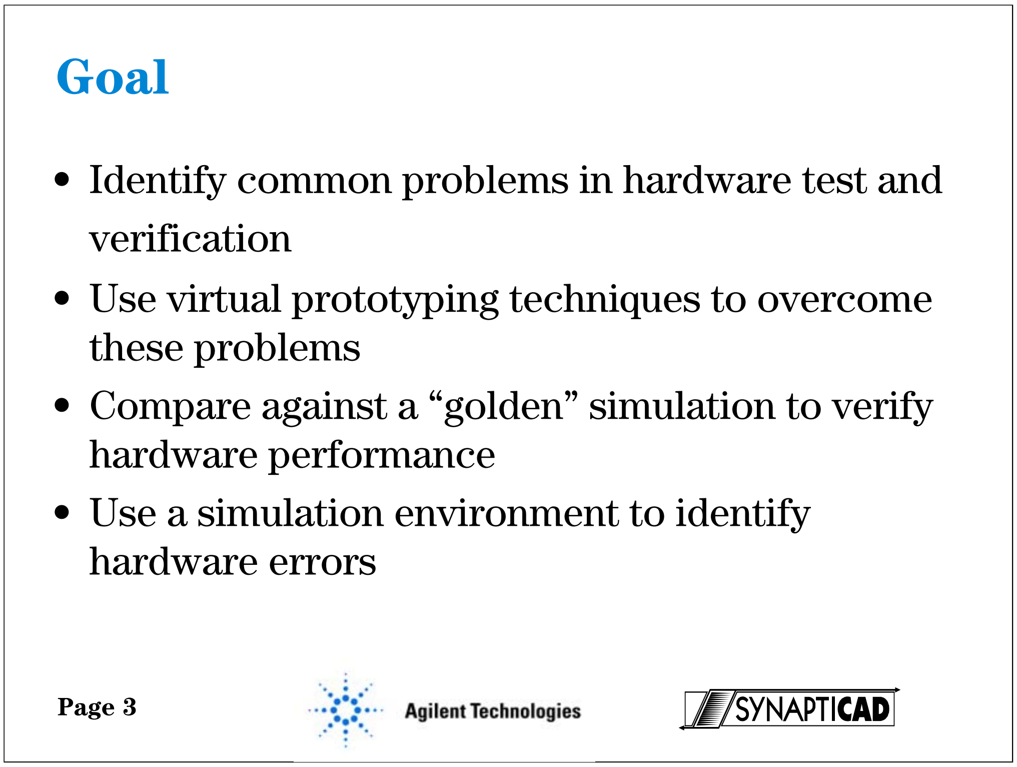
- Demonstrate how EDA tools and traditional test and measurement tools can synergistically work together to reduce the time for design and debug by effectively combining the worlds of simulation and physical measurement.

Agilent Technologies

SYNAPTICAD

If you have been reading some technical publications you would start to believe that the world of EDA tools and the world of traditional debug are mutually exclusive.

To some extent this has been true. But it doesn't need to be. In fact today's objective is to look at ways of combining the strengths of both methodologies into a single strategy that will reduce the design cycle, make more efficient use of your time and get a higher quality product to market sooner.



Goal

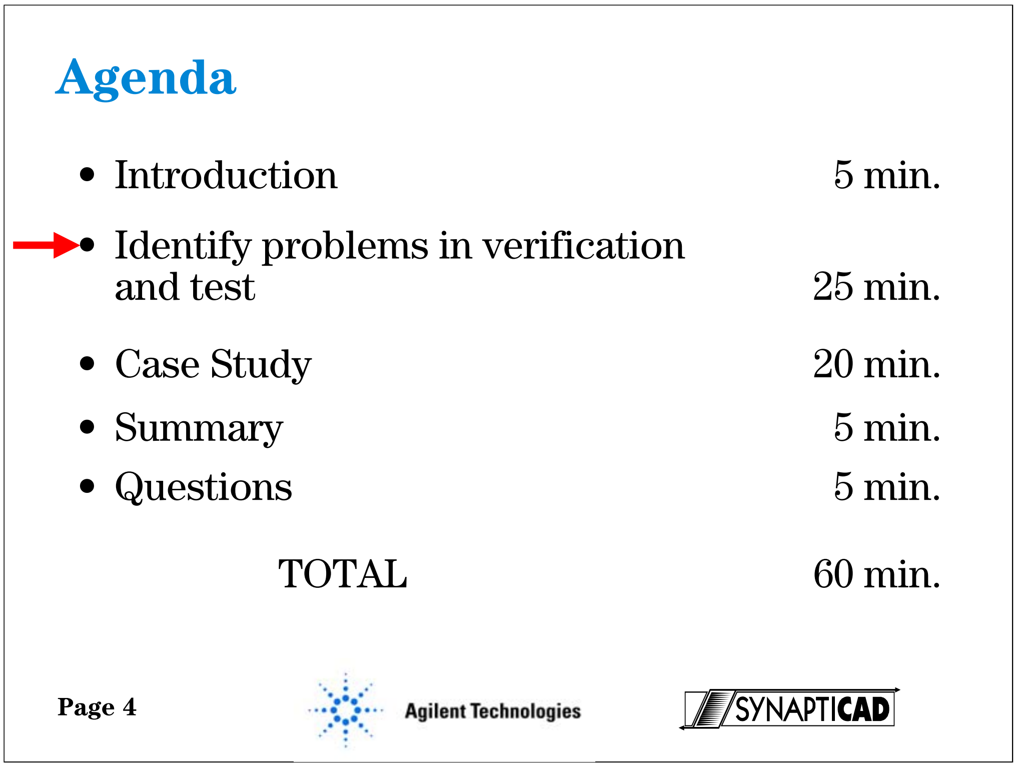
- Identify common problems in hardware test and verification
- Use virtual prototyping techniques to overcome these problems
- Compare against a "golden" simulation to verify hardware performance
- Use a simulation environment to identify hardware errors

Agilent Technologies

SYNAPTICAD

Hardware test and verification is a costly and time-consuming part of the development of digital hardware. Virtual prototyping techniques using a combination of pattern generators, logic analyzers, and EDA software can leverage the work done during the design phase of the product, simplifying the development of a test environment that provides good test coverage and excellent debug capability.

In this paper we will examine problems that have limited the use of virtual prototyping and explore simple, easy-to-use methods for overcoming these problems. We will further demonstrate additional benefits that result when virtual prototyping techniques are combined with the analysis and troubleshooting capabilities of a simulation environment.



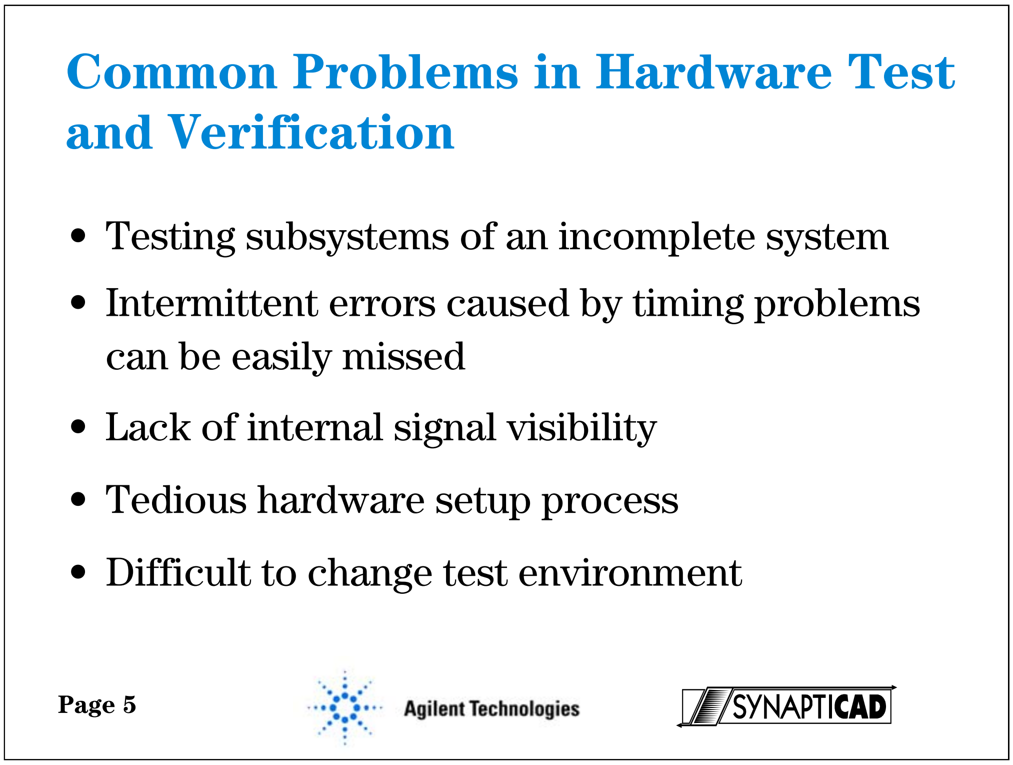
Agenda

• Introduction	5 min.
• Identify problems in verification and test	25 min.
• Case Study	20 min.
• Summary	5 min.
• Questions	5 min.
TOTAL	60 min.

Agilent Technologies

SYNAPTICAD

This presentation will last 1 hour.



Common Problems in Hardware Test and Verification

- Testing subsystems of an incomplete system
- Intermittent errors caused by timing problems can be easily missed
- Lack of internal signal visibility
- Tedious hardware setup process
- Difficult to change test environment

Page 5 Agilent Technologies SYNAPTICAD

Thank you Gregg. My name is Peter Menegay, and I'm an engineer with Synapticad. If you could turn now to slide 5, I'd like to discuss some of the common problems that have traditionally plagued engineers doing hardware test and verification. I'm sure many of you are familiar with these points, but I'd like to do this in order to provide a launch pad for discussing our proposal for solving these problems, or at least ameliorating them, which is built around this idea that Gregg mentioned of virtual prototyping.

These problems generally boil down to, first of all,

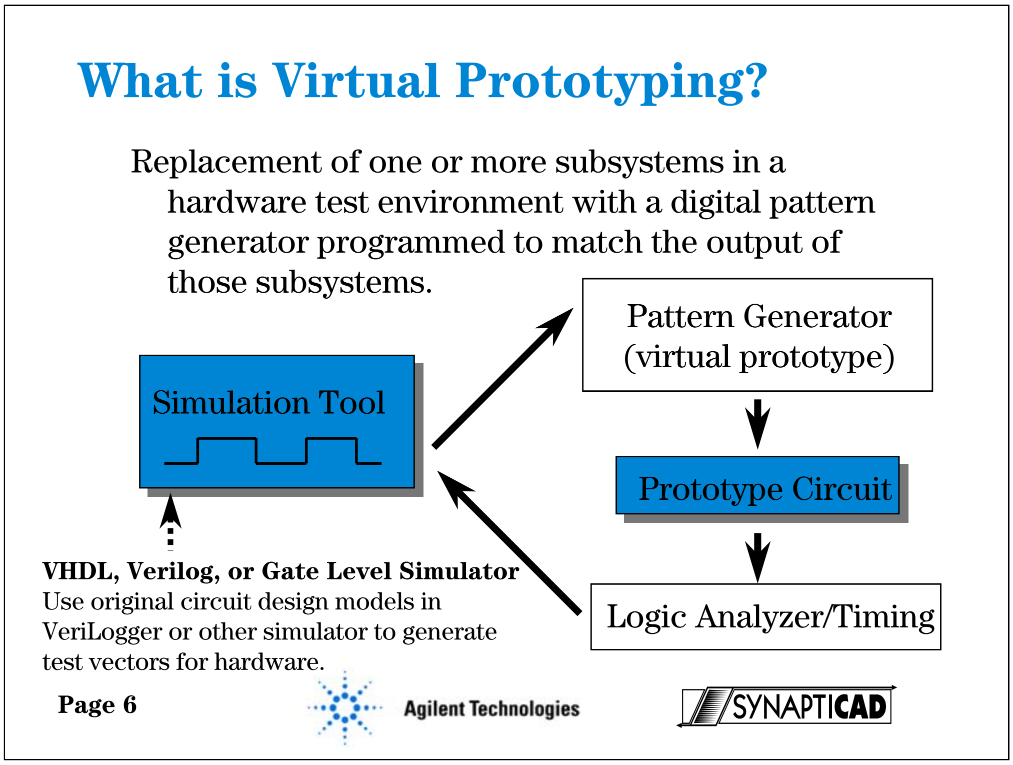
(1) The need to test a subsystem before a complete version of the system is available.

(2) Detecting intermittent errors due to timing problems that don't necessarily manifest themselves during a given test run. This happens when timing violations exist within the circuit but they are not manifesting themselves because of some "favorable" condition, such as temperature, or voltage variation which suppresses the error. Hence the circuit seems to work correctly during the test, but is really hiding a timing violation which could come out later. We would, obviously, like to know about these kinds of violations.

(3) Difficulty with debugging because the test environment doesn't have access to all internal circuit nodes.

(4) Test systems that often don't have the capability to isolate a fault to a single subsystem.

(5) The tedious setup of the hardware test environment (signal names, clock speeds, etc.) because design-level information can't be directly imported from EDA tools. This typically involves an engineer sitting in front of a pattern generator and laboriously programming it, and then sitting in front of a logic analyzer and visually trying to figure out, by looking at the waveforms, whether the circuit is behaving properly. Needless to say, this is a tough assignment, and one which is fraught with error.



What is Virtual Prototyping?

Replacement of one or more subsystems in a hardware test environment with a digital pattern generator programmed to match the output of those subsystems.

VHDL, Verilog, or Gate Level Simulator
Use original circuit design models in VeriLogger or other simulator to generate test vectors for hardware.

Page 6

Agilent Technologies SYNAPTICAD

If you'll turn to the next slide, slide 6,

Let's talk about what we mean by virtual prototyping. Virtual prototyping means the combination of the traditional test methods using a pattern generator/logic analyzer with a software simulation tool. The traditional method consists of programming the pattern generator with test vectors for input to the prototype circuit, and then analyzing the response of the circuit by importing it into a logic analyzer. The EDA enhanced method is to use a simulation tool to generate the test vectors, export them to the pattern generator, perform the test on the prototype, and then import the response signals from the logic analyzer back into the same simulation tool. We are going to see that the use of this software environment significantly enhances the test process by reducing the time required to perform the work, and by increasing the quality and reliability of the results.

It is important to note here that the simulation tool is capable of generating stimulus based on a VHDL or Verilog simulation, either done internally or imported from a third party simulator. Also, this tool is capable of taking VHDL or Verilog testbenches and turn them into pattern generator stimulus. Combining this with the ability to read the output of a logic or timing analyzer provides a very powerful yet easy to use environment for hardware test and debug.

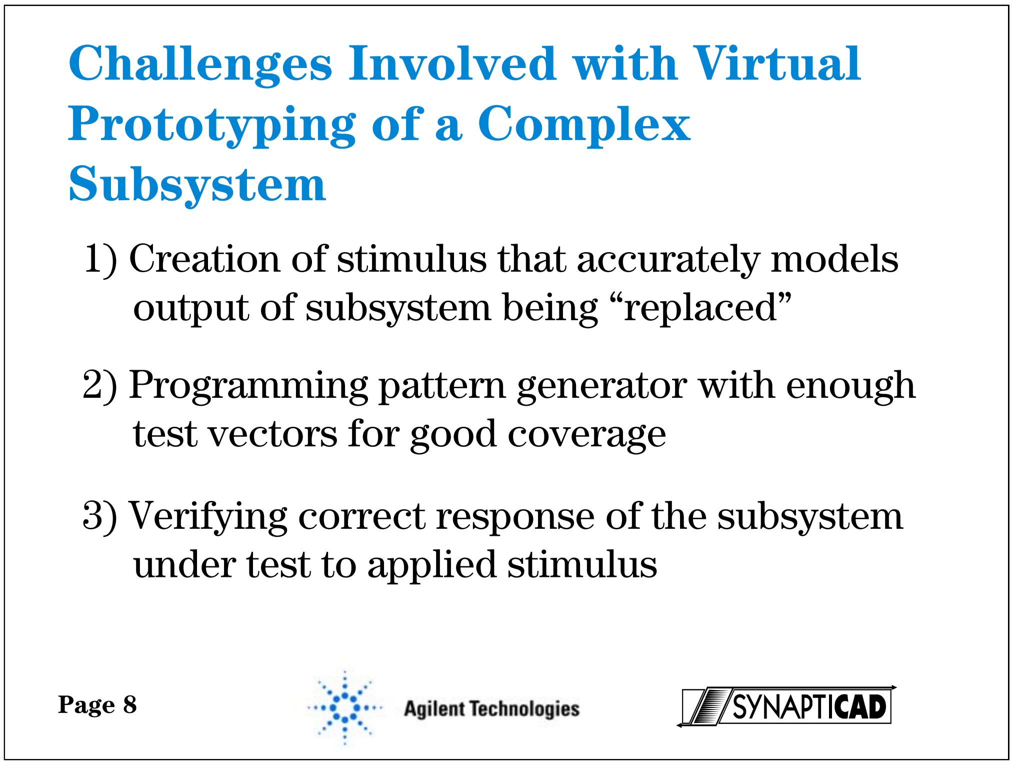
During the development of a complex digital system, individual subsystems often get prototyped at different times. With virtual prototyping techniques, these subsystems can be tested in the absence of other subcomponents using general-purpose programmable pattern generators to emulate the unfinished parts of the system (in other words, the pattern generator serves as a virtual prototype).

What is the difference between Emulation and Virtual Prototyping?

Benefits of Virtual Prototyping

- Reuse of simulation stimulus and response
- Early testing of subsystems
- Easier to identify bugs in isolated subsystems
- Virtual prototypes are easy to modify
- Easy to characterize speed of subsystem

Virtual prototyping solves many of the problems faced during creation of a robust hardware test environment and provides additional debugging capabilities as well: (1) Since data has already been generated during the design process (in simulation) this data can automatically be used as stimulus/or to check against. (2) subsystems can be tested before the whole system is complete, (3) subsystems can be tested in isolation to narrow down the location of bugs and simplify the scope of the debug effort, (4) virtual prototypes can easily be modified to further debug the subsystem under test in the event that design flaws are found in the virtual prototype or to help study the nature of the problem with the subsystem under test, and (5) the speed performance of a subsystem can easily be characterized by increasing the clock frequency at which the stimulus from the virtual prototype is applied.



Challenges Involved with Virtual Prototyping of a Complex Subsystem

1) Creation of stimulus that accurately models output of subsystem being “replaced”

2) Programming pattern generator with enough test vectors for good coverage

3) Verifying correct response of the subsystem under test to applied stimulus

Virtual prototyping is not a new idea, of course, as pattern generators were created for just this purpose, but several practical issues have limited the complexity of the systems that could be virtually prototyped. Creating complex stimulus that accurately models the environment surrounding a subsystem and programming the stimulus into the pattern generator have, until recently, been engineering-intensive tasks.

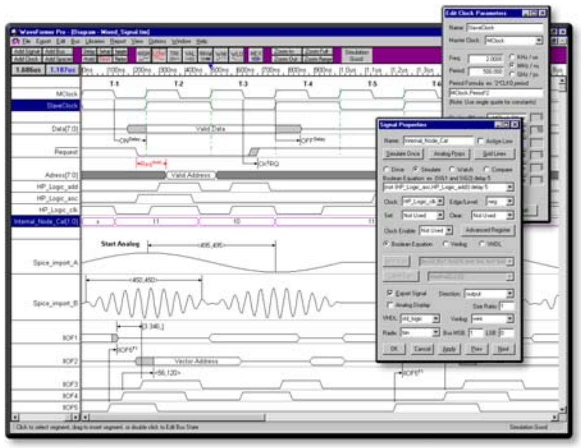
An additional challenge using virtual prototypes has been to verify the response of the hardware being tested. The response of the system being tested has to be verified in any kind of hardware verification procedure whether using virtual prototypes or with a complete system test, but a more thorough verification process should be used with a virtual prototypes since they don’t react to the output of the system under test (when tested in a complete system the other subcomponents help provide an indirect degree of test coverage since they often depend on the output of the subsystem under test in order to perform correctly themselves).

NOTE:

Test coverage means how much of circuit we are able to test. It is applying enough different tests to get the circuit fully checked. So if simulation performed enough tests, then automatically the test coverage is adequate.

1) Creation of Stimulus Vectors

Provide a graphical user interface and the ability to import simulation vectors to ease the task of creating and editing of complex stimulus.



Agilent Technologies

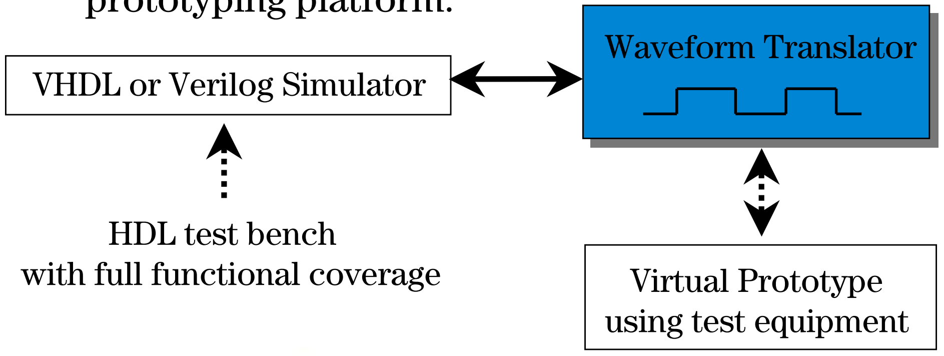
SYNAPTICAD

SynaptiCAD's WaveFormer and VeriLogger products offer a timing diagram editing environment that enables stimulus to be created using a combination of graphically drawn signals, timing parameters that constrain edges, clock signals, and temporal and Boolean equations for describing complex, quasi-repetitive signal behavior. Advanced operations on signals such as time scaling and shifting, and block copy and pasting of signal behavior over an interval of time are also supported. This simple, but powerful environment dramatically eases the labor associate with the generation of complex stimulus.

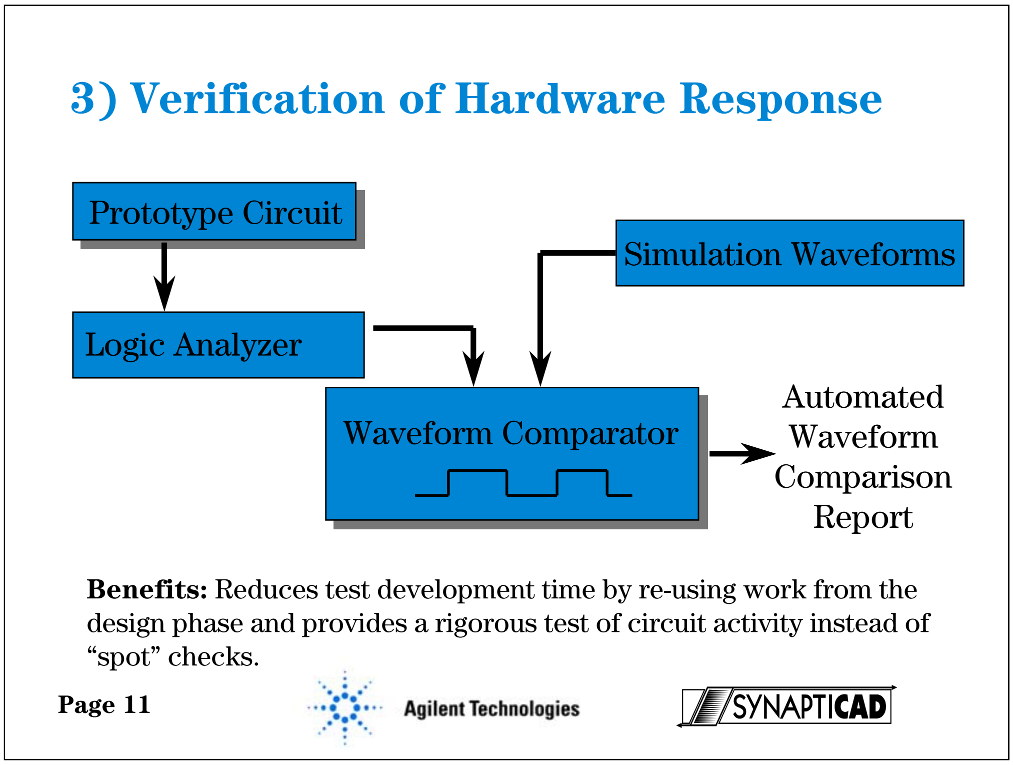
Stimulus can also be created from design simulation waveforms or even from real world data acquired by a logic analyzer. All the above-mentioned manipulations can be performed on these waveforms as well. For example, assume a set of waveform data was gathered from a current generation system running at 50Mhz and the new (not yet completed) system will run at 90Mhz. The captured waveforms can be scaled to the higher speed and then converted to pattern generator stimulus to test the completed portions of the new system.

2) Test Vector Coverage

Use simulation test vectors with full functional coverage to drive the hardware prototyping platform.



Reuse complex stimulus vectors developed during simulation verification as hardware stimulus vectors. This ensures that if adequate test vectors were created during simulation verification, you will have adequate functional coverage during hardware verification.



3) Verification of Hardware Response

Prototype Circuit → Logic Analyzer → Waveform Comparator ← Simulation Waveforms

Waveform Comparator → Automated Waveform Comparison Report

Benefits: Reduces test development time by re-using work from the design phase and provides a rigorous test of circuit activity instead of "spot" checks.

Page 11   Agilent Technologies   SYNAPTICAD

Hardware response verification comprises two basic steps:

1) Capture waveforms from circuit using logic analyzer and import into an automated waveform comparitor.

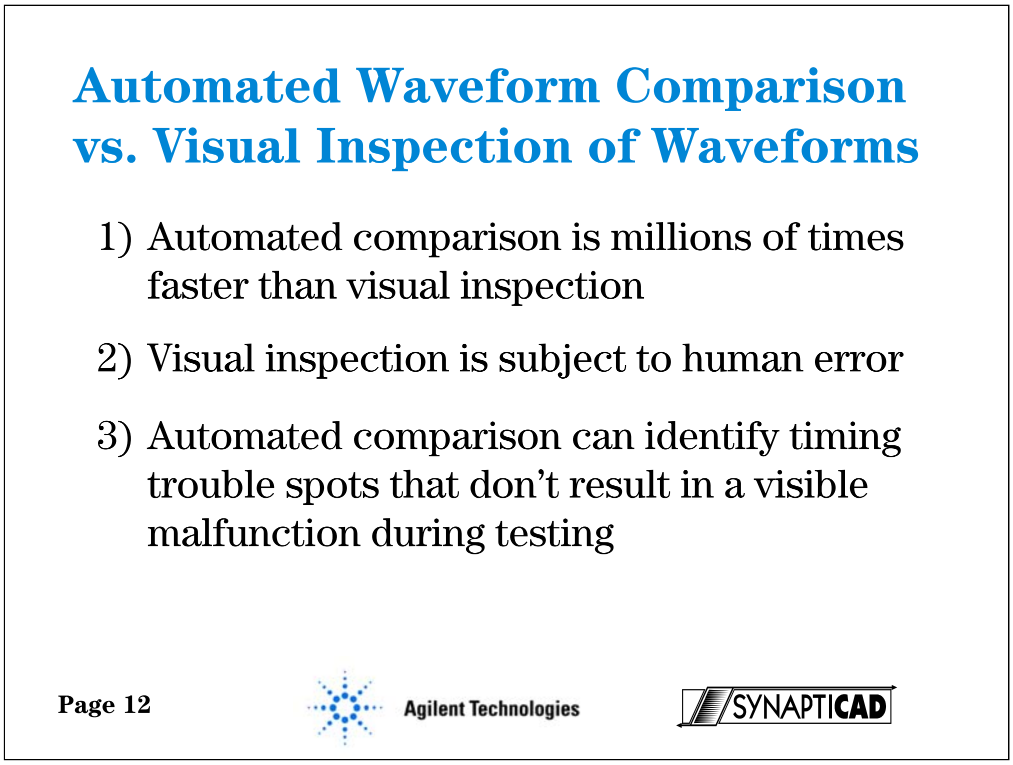
2) Generate a comparison report against simulation results to verify circuit operation.

There are two kinds of critical comparisons between waveforms: edge-by-edge comparisons and clocked comparisons. An edge-by-edge comparison shows every difference between the compared waveforms and is generally used when comparing asynchronous waveforms or looking for subtle timing errors. Clocked comparisons check for waveform differences slightly before and after clock edges. They are useful for verifying synchronous circuits because they avoid reporting differences that result because of glitches and slight differences in delay times, events which generally don't cause errors in a synchronous circuit.

Negative and positive time windows can be specified around the clock edge of a clocked comparison to verify that register and latch timing constraints are being met. This is useful in identifying potential timing errors that generate intermittent errors that may not show up directly during hardware testing. Specifying a negative window time helps to catch setup errors that arise when delay paths through the circuit are too long. Specifying a positive window time helps to catch hold errors that arise when delays paths are too short. These errors often go uncaught in initial system tests because they vary with operating conditions such as temperature and supply voltage. Using time windows to verify stability around the clock edges, potential trouble spots can quickly be identified and investigated.

Negative and positive time windows can be specified around the clock edge of a clocked comparison to verify that register and latch timing constraints are being met.

This is useful in identifying potential timing errors that generate intermittent errors that may not show up directly during hardware testing. Specifying a negative window time helps to catch setup errors that arise when delay paths through the circuit are too long. Specifying a positive window time helps to catch hold errors that arise when delay paths are too short. These errors often go uncaught in initial system tests because they vary with operating conditions such as temperature and supply voltage. Using time windows to verify stability around the clock edges, potential trouble spots can quickly be identified and investigated.



Automated Waveform Comparison vs. Visual Inspection of Waveforms

1) Automated comparison is millions of times faster than visual inspection
2) Visual inspection is subject to human error
3) Automated comparison can identify timing trouble spots that don't result in a visible malfunction during testing

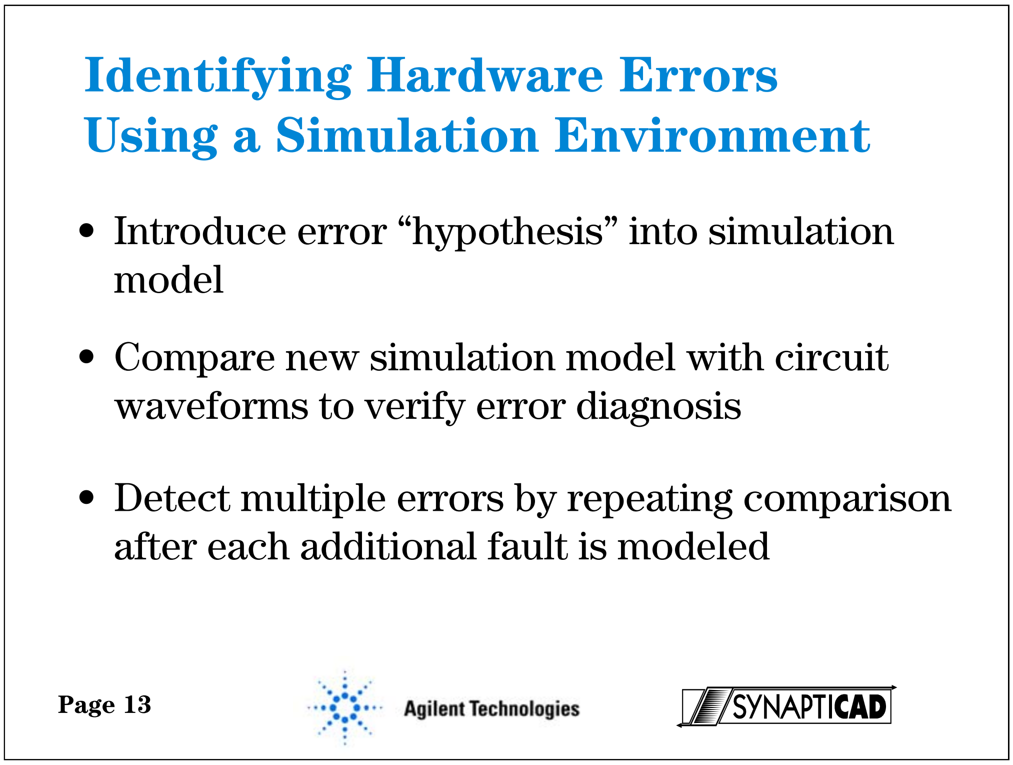
Agilent Technologies

SYNAPTICAD

Traditionally, waveform data from a logic analyzer has required visual inspection by an engineer familiar with the operation of the circuit to verify proper operation or to troubleshoot an error. This method is error-prone and time-consuming, especially as the number of waveforms and the amount of data captured increases. Automated comparison guarantees a rigorous check of each data point, ensuring the detection of "small impact" errors that are easily missed during visual inspection of the waveforms.

Another problem with visual inspection of waveforms is that even when an error is spotted, there is no guarantee that the source of the error didn't happen earlier in the waveform data set and was simply missed during visual inspection. This can lead to a slow hunt back in time through the waveform data to the original divergence from correct operation. With automated comparison, the original divergence is immediately detected, speeding the debug process.

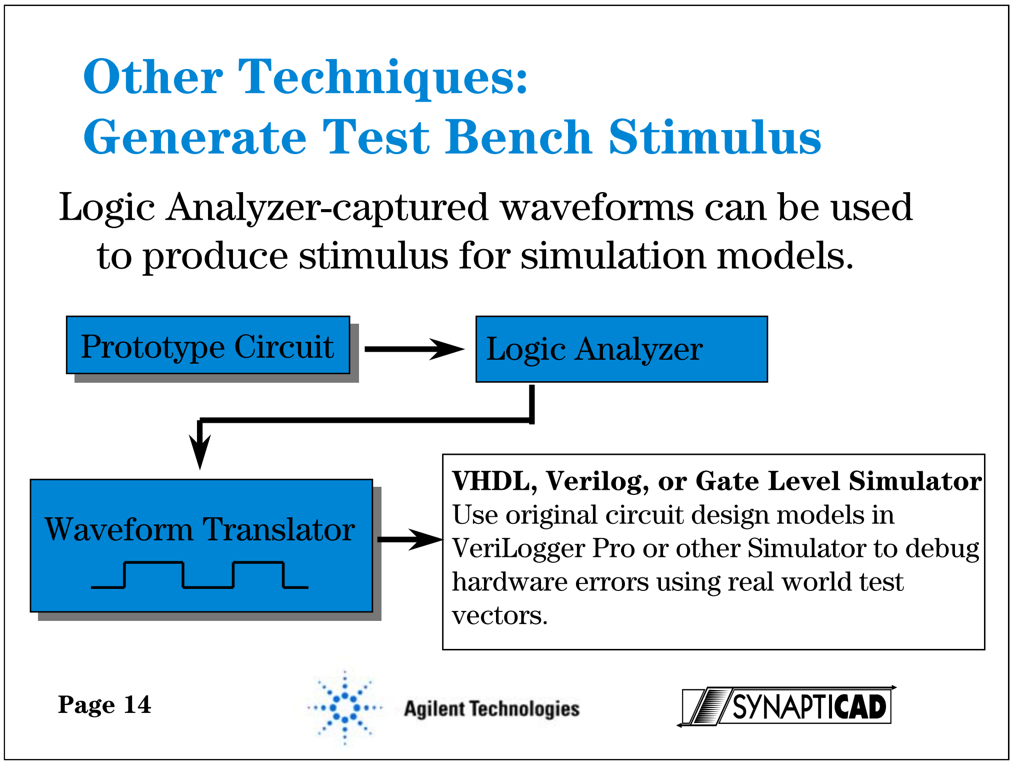
Another important advantage of automated verification is it reduces the amount of knowledge required by the verification engineer to test the system. Rarely does even the designer of a system keep a detailed vision of the operation of all the signals in his design (that's the reason for simulators), yet that is exactly the capability needed to spot a hardware error. The simulation environment must "know" exactly how the signals should be acting and "inspect" the waveforms, identify the faulty signal, and display the difference between the actual response and the correct response.



Identifying Hardware Errors Using a Simulation Environment

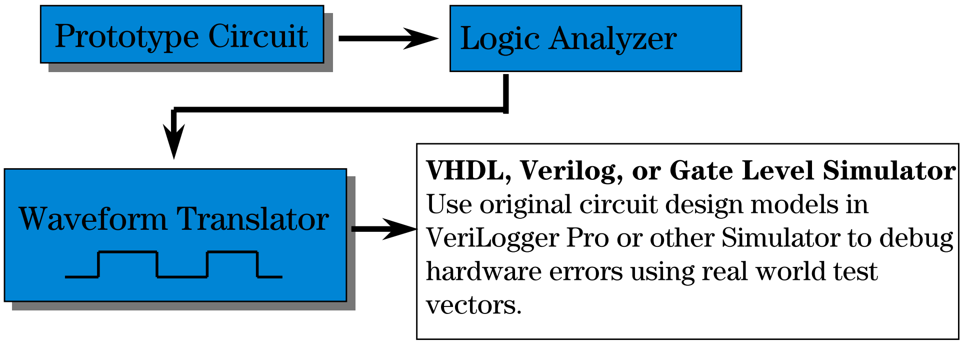
- Introduce error “hypothesis” into simulation model
- Compare new simulation model with circuit waveforms to verify error diagnosis
- Detect multiple errors by repeating comparison after each additional fault is modeled

One of the more powerful debug capabilities associated with verifying the hardware results using a simulation environment is the ability to check that the suspected circuit fault would generate the observed results. This can be done by adding the suspected fault to the simulation model, re-running the simulation, and comparing the new simulation results to the captured waveforms. If the waveforms match, the error has most likely been correctly identified. If there are still mismatches further downstream in the waveform data, this is likely an indication that the error has been either misidentified or that the hardware contains multiple errors. Repetition of the above process after identifying each error can reduce the number of times the hardware needs to be changed.



Other Techniques: Generate Test Bench Stimulus

Logic Analyzer-captured waveforms can be used to produce stimulus for simulation models.

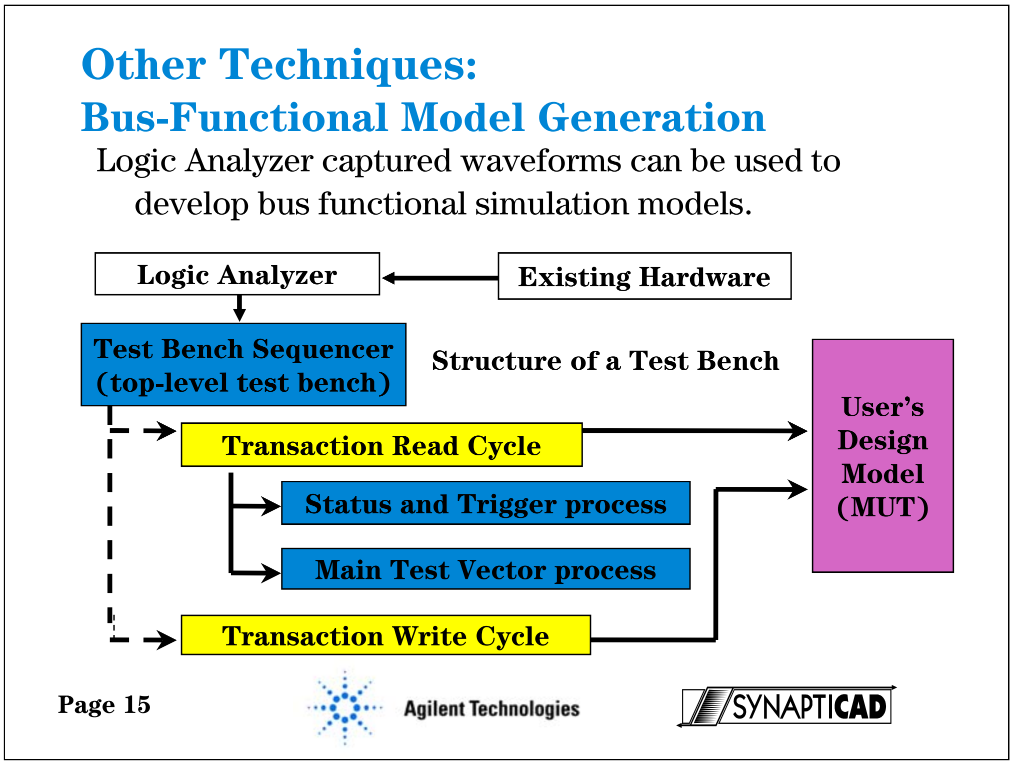


VHDL, Verilog, or Gate Level Simulator
Use original circuit design models in VeriLogger Pro or other Simulator to debug hardware errors using real world test vectors.

Agilent Technologies SYNAPTICAD

Although this talk has focused on using design data to help verify hardware, the reverse process can also be successfully applied to the design and simulation of new systems. Most systems being designed need to interface with already existing hardware (IC's or entire boards) and simulation models are frequently not available for that hardware. Waveforms from the existing hardware need to be captured with a logic analyzer and converted to HDL test bench code or SPICE stimulus.

Let me give you a quick example of a customer who has used this technique. A customer from Microwave Data Systems was testing an ASIC design in which he needed to generate stimulus. Instead of programming these manually into his VHDL simulator he simply captured output from his upstream components using the simulation software tool(which he had), fed them into a VHDL simulator which he had programmed, and produced the needed stimulus for output to his new design. He reported that the process took about 15 minutes rather than an estimated 2 weeks. This is a real world example from a customer, who without any prompting by us, volunteered this information. Needless to say, these new techniques for testing are going to provide a real productivity boost.



Other Techniques: Bus-Functional Model Generation

Logic Analyzer captured waveforms can be used to develop bus functional simulation models.

Logic Analyzer
Existing Hardware
Test Bench Sequencer (top-level test bench)
Structure of a Test Bench
Transaction Read Cycle
Status and Trigger process
Main Test Vector process
Transaction Write Cycle
User's Design Model (MUT)

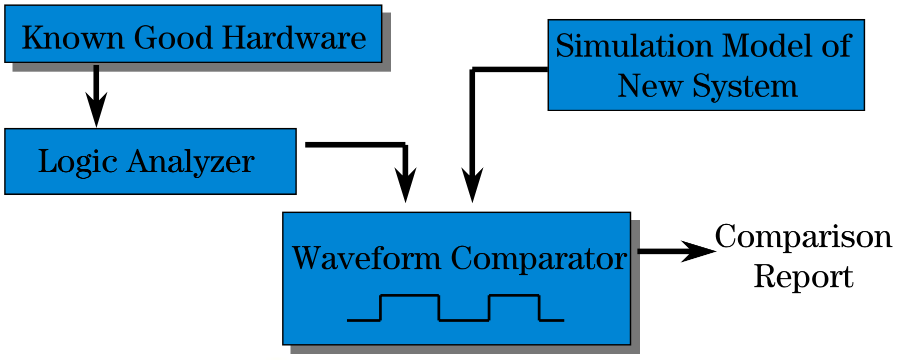
Agilent Technologies SYNAPTICAD

Another feature is the ability to verify that a system being designed can work with an existing piece of hardware. In this case, we use the logic analyzer to capture data from the existing system, where it can be imported into a test bench generation tool where we can massage the waveforms to create a reactive, bus functional model of the existing system. We then use this model to stimulate the simulation model of our new design and to verify it's response. This approach is similar to the technique on the previous slide, but in this case we break up the captured data into reusable "transactions". You can then control the order in which these transactions are applied to your model under test with function calls in the top-level of your test bench. An example of a transaction would be a write cycle or a read cycle by a microprocessor. These transactions can be parameterized in the testbench generation tool by replacing specific data values with variables which can be passed in when you call transaction function call.

When simulating a design using an HDL language, the waveforms should be converted directly to test vectors or used as a starting point by the designer for the creation of a reactive, bus functional model capable of checking and responding to the output of the system being simulated.

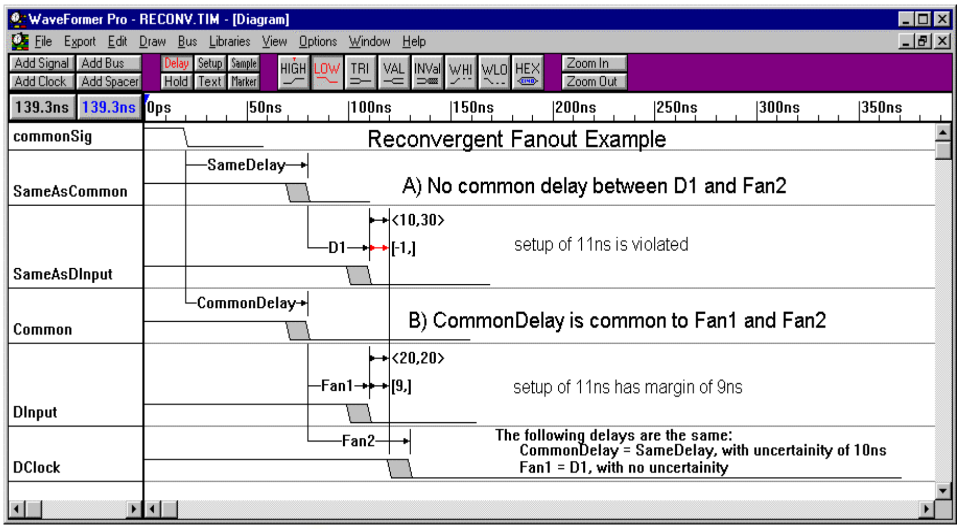
Other Techniques: Interface Functional Checking

Compare waveforms from “known good” hardware against next generation model



Often the system being designed is the next generation of an existing product with similar functionality. In this case, the new system must generally mimic at least part of the interface of the older system. By performing a waveform comparison between the old system and the new design, correct functioning of the new system can be assured.

Document Cause-Effect Relationships


WaveFormer Pro - RECONV.TIM - [Diagram]
File Export Edit Draw Bus Libraries View Options Window Help
Add Signal Add Bus Add Clock Add Spacer
Delay Setup Sample Hold Text Marker
HIGH LOW TRI VAL INVal WHI WLO HEX
Zoom In Zoom Out
139.3ns 139.3ns
0ps 50ns 100ns 150ns 200ns 250ns 300ns 350ns
commonSig
SameAsCommon
SameAsDInput
Common
DInput
DClock
SameDelay
D1
CommonDelay
Fan1
Fan2
<10,30>
[-1,]
<20,20>
[9,]

Reconvergent Fanout Example

A) No common delay between D1 and Fan2

setup of 11ns is violated

B) CommonDelay is common to Fan1 and Fan2

setup of 11ns has margin of 9ns

The following delays are the same:
CommonDelay = SameDelay, with uncertainity of 10ns
Fan1 = D1, with no uncertainity

Agilent Technologies

SYNAPTICAD

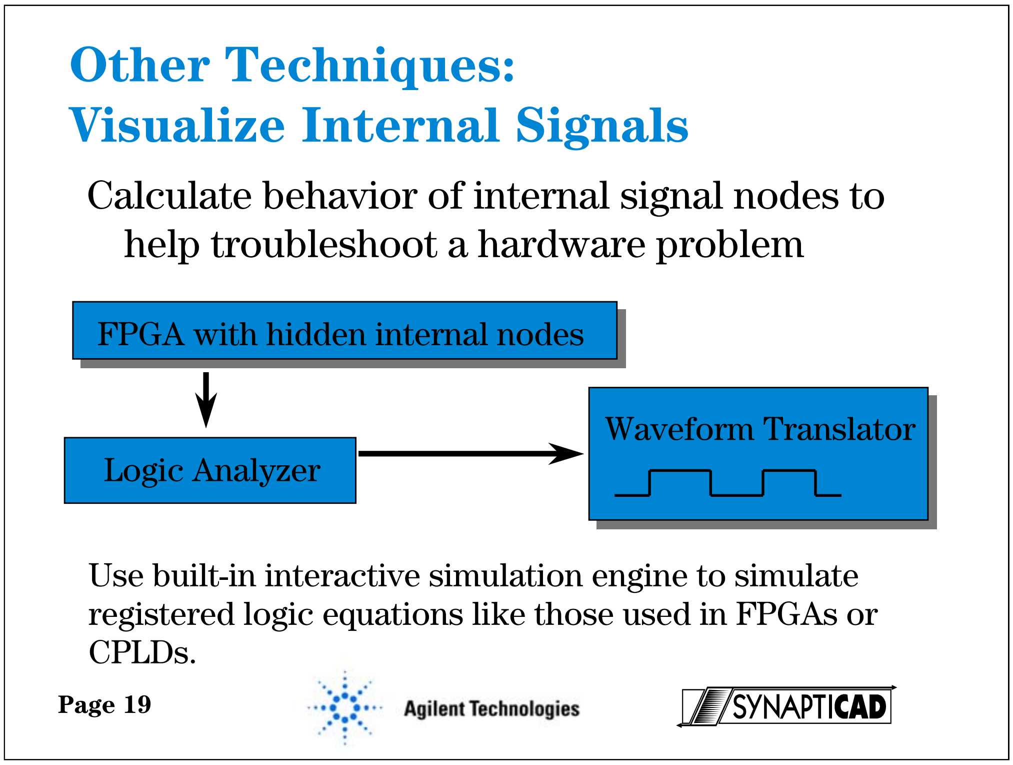
It is important to document circuit operation by converting captured waveform data into true timing diagrams. The user can add delay, setup, and hold timing parameters to document the temporal relationships between signal transitions. Finally, improvements to the readability of timing diagrams can be obtained by adding text and grid lines. When the timing diagram is complete, WaveFormer Pro can create publication quality WMF, MIF or EPS images for use in Word, FrameMaker or PDF files.

Other Techniques: Continuous Setup and Hold Checking

- Perform a post analysis of the circuit waveforms to check for setup and hold violations. Perform a check on each clock edge and generate a time-ordered report of violations of the timing constraints entered by the user.

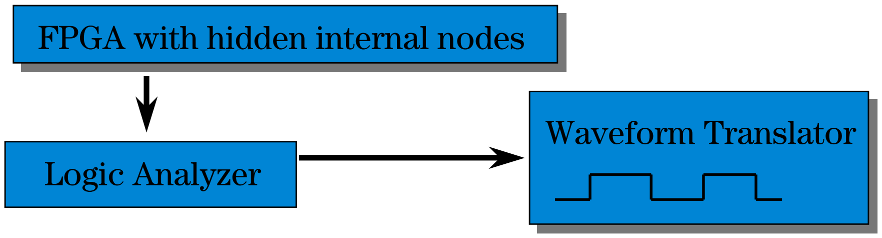
The simulation environment should be able to generate a report of ALL setup and hold timing violations specified between any two signals in a timing diagram regardless of whether the signal is a captured waveform or a simulated waveform (logic analyzers typically only flag the first violation).

Setup and hold time violations in ASICs and PLDs are particularly troublesome because timing violations usually occur on flip-flop inputs that are not directly available at device pins, but are instead a logical function of the device's inputs. Using conventional debugging techniques, these timing violations are extremely difficult to catch because they cannot be directly measured. WaveFormer's ability to simulate internal signals makes it simple to detect timing violations between signals buried inside a chip.



Other Techniques: Visualize Internal Signals

Calculate behavior of internal signal nodes to help troubleshoot a hardware problem



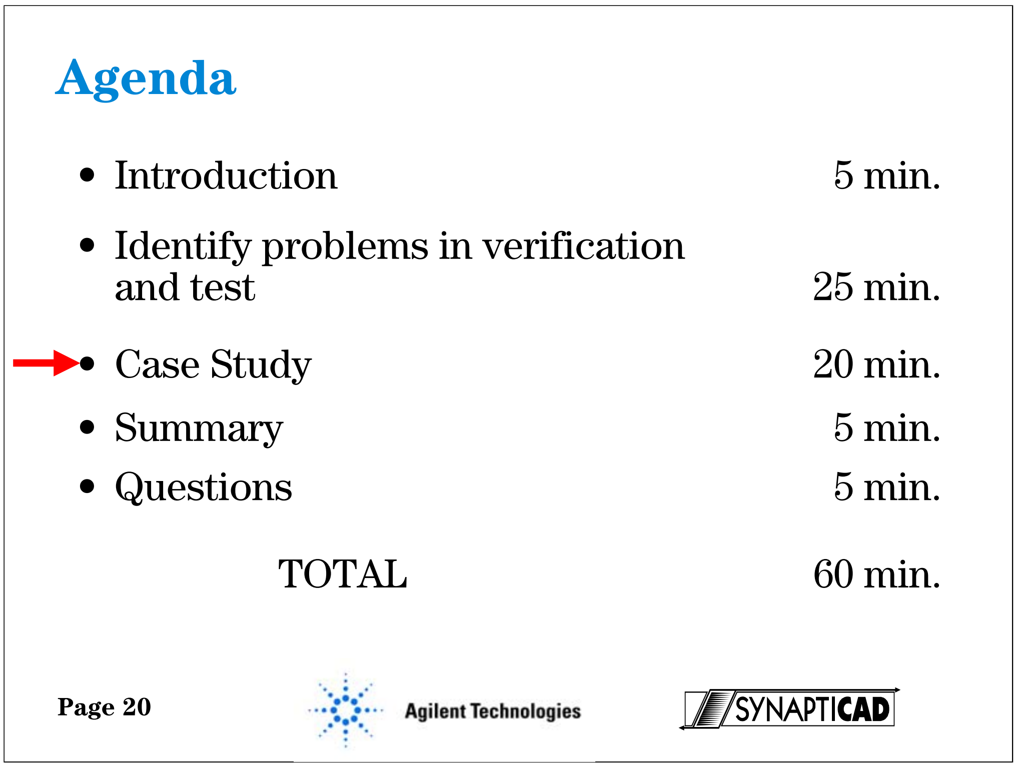
Use built-in interactive simulation engine to simulate registered logic equations like those used in FPGAs or CPLDs.

Agilent Technologies

SYNAPTICAD

One of the most frustrating problems encountered when debugging a circuit is the inability to see what is happening on all the internal signal nodes of an FPGA or ASIC. A logic analyzer can only show the activity on signals that are brought out on device pins. Unfortunately, many designs are I/O limited. Even when there are no limitations, there are almost never enough pins available to bring out all the useful nodes.

To combat this problem, use a built-in interactive simulation engine that can simulate registered logic equations like those used in FPGAs or CPLDs. By combining this capability with data captured by a logic analyzer, users can determine what is happening not only at the pins of their devices, but also on the internal nodes that cannot be directly probed. This dramatically simplifies the debug process by allowing a designer to effectively trace into a chip to locate the source of the problem.

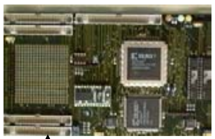


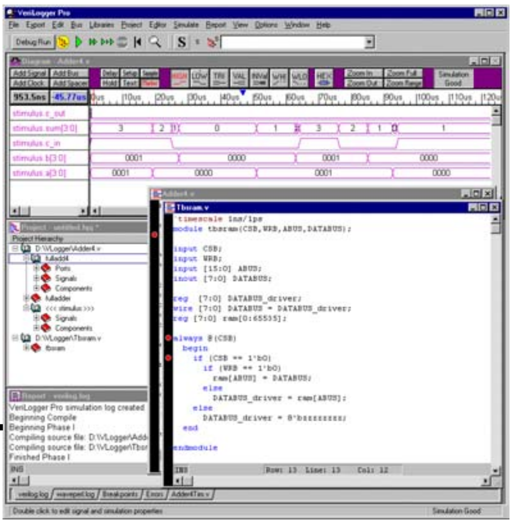
Agenda

• Introduction	5 min.
• Identify problems in verification and test	25 min.
→ • Case Study	20 min.
• Summary	5 min.
• Questions	5 min.
TOTAL	60 min.

If you'll turn now to slide 20, this concludes this part of the presentation where I've detailed some of the problems with hardware test and verification and proposed a solution based on the combination of EDA software and the traditional methods. I'd like to emphasize that what we've done here is replaced a difficult to modify hardware test environment with an easy to modify software environment, without sacrificing the "realness", if you will, of hardware testing. Now I'd like to turn it back over to Gregg who's going to provide us with a case study so you can see an example of how these ideas I've been talking about really work.

Case Study of Virtual Prototyping





```
`timescale 1ns/1ps
module tbsram(CSB,WRB,ABUS,DATABUS);

input CSB;
input WRB;
input [15:0] ABUS;
inout [7:0] DATABUS;

reg  [7:0] DATABUS_driver;
wire [7:0] DATABUS = DATABUS_driver;
reg [7:0] ram[0:65535];

always @(CSB)
  begin
    if (CSB == 1'b0)
      if (WRB == 1'b0)
        ram[ABUS] = DATABUS;
      else
        DATABUS_driver = ram[ABUS];
    else
      DATABUS_driver = 8'bzzzzzzzz;
  end

endmodule
```

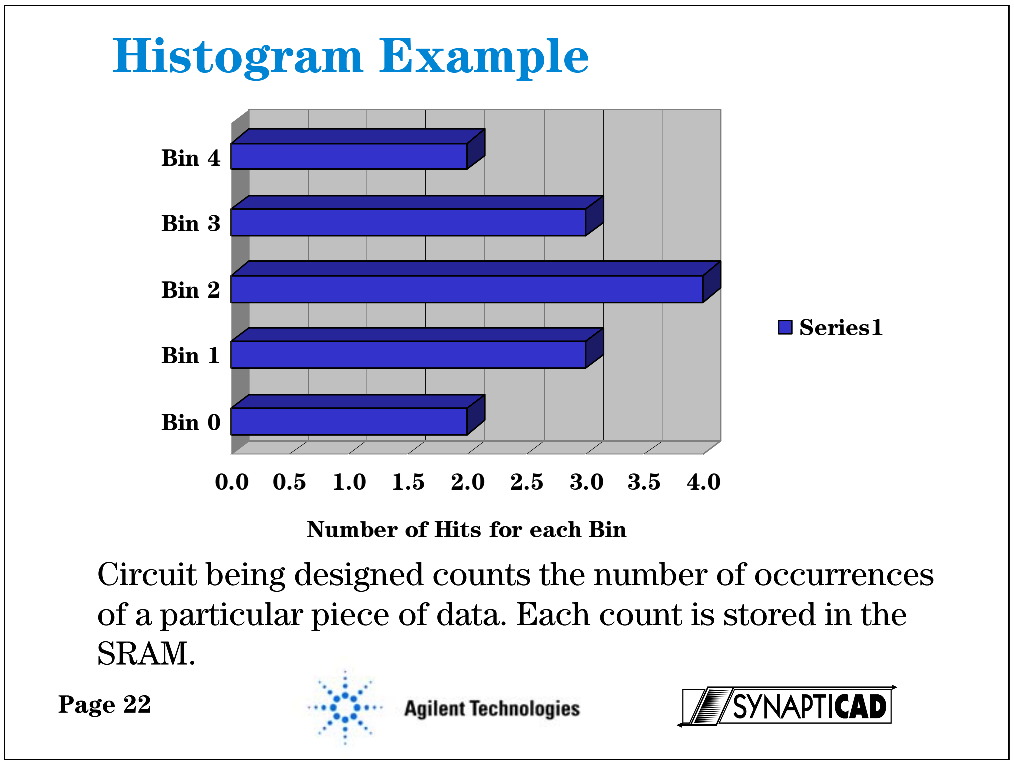
```
VeriLogger Pro simulation log created
Beginning Compile
Beginning Phase I
Compiling source file: D:\VLogger\Add
Compiling source file: D:\VLogger\Tbsr
Finished Phase I
```

Agilent Technologies

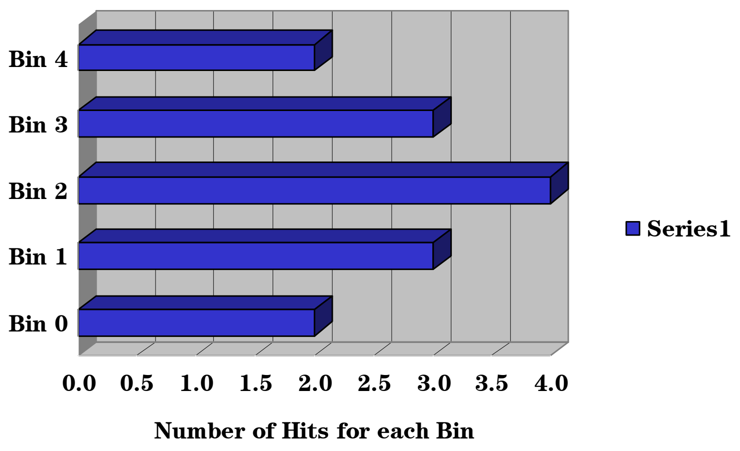
SYNAPTICAD

To better understand the processes described in this paper the following case study is provided.

A circuit design will be developed using SynaptiCAD and Xilinx design tools. The design will then be downloaded onto an Associated Professional Systems X240 development board. This board uses a Spartan XS30 FPGA and 256K of SRAM. An Agilent Technologies 16522A pattern generator will provide stimulus to the device under test and the results will be measured with an Agilent Technologies 16717A timing analyzer. The results will then be uploaded to the SynaptiCAD simulator and the results will be automatically compared.



Histogram Example

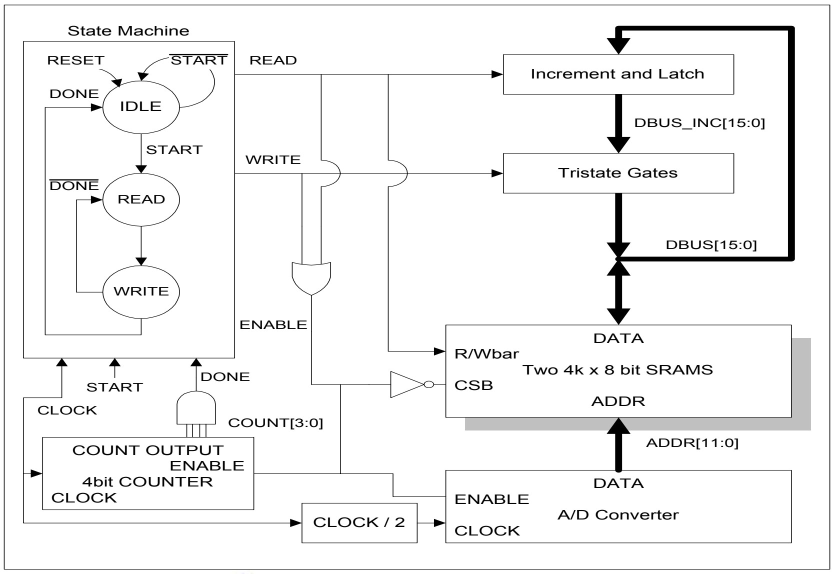


Circuit being designed counts the number of occurrences of a particular piece of data. Each count is stored in the SRAM.

Agilent Technologies SYNAPTICAD

As an example application of the use of a virtual prototype, we will examine the design and debug of a histogram circuit that is used for A/C performance testing of analog-to-digital converters. The histogram circuit counts the occurrences of 12-bit values received from the Analog to Digital Converter (ADC). These "bin" counts can be graphed on a bar chart to show the relative frequency of occurrence of each data value (a histogram chart).

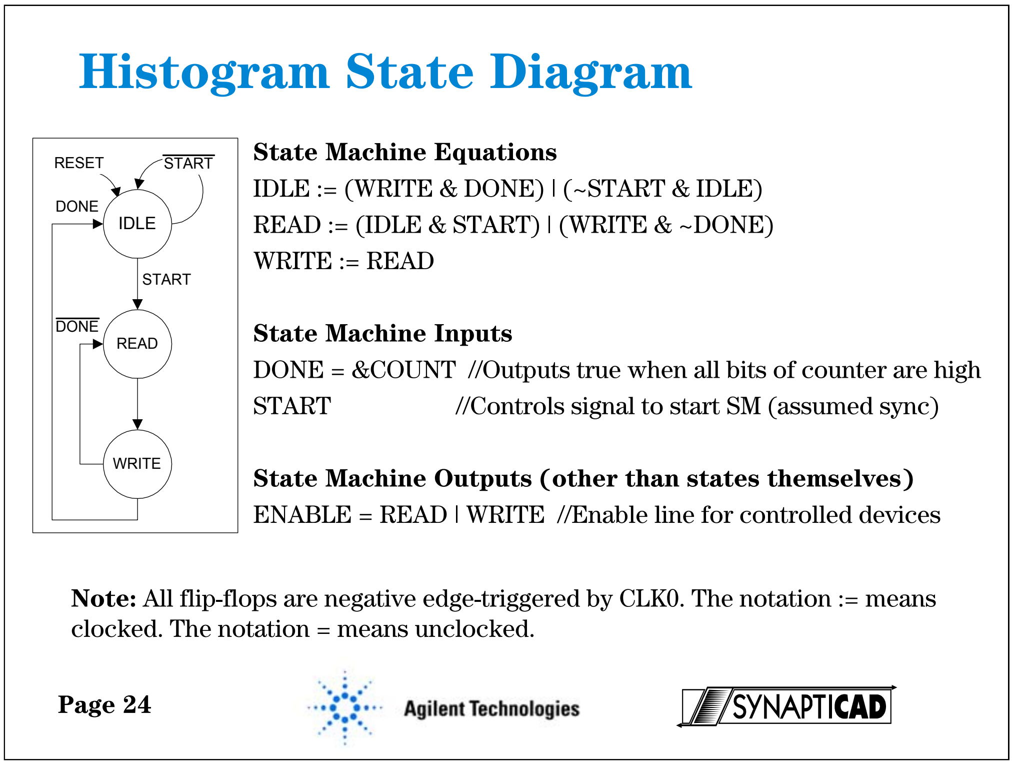
Block Diagram of Histogram



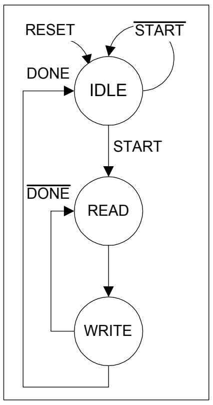
Agilent Technologies SYNAPTICAD

The histogram circuit is composed of three subsystems: the ADC which generates the data to be histogrammed, the data path circuit that stores and increments the bin count for each value, and the state machine that generates the control signals for the data path circuit.

In this design the ADC will not be created. Instead the Agilent Technologies pattern generator will provide test vectors to simulate the 12 bit output. The purpose in not creating the ADC is to demonstrate that when part of a design is not available it is still possible to test and debug the design. This not only saves time, but by testing smaller pieces it becomes easier to find problems when they do occur.



Histogram State Diagram



State Machine Equations

IDLE := (WRITE & DONE) | (~START & IDLE)
READ := (IDLE & START) | (WRITE & ~DONE)
WRITE := READ

State Machine Inputs

DONE = &COUNT //Outputs true when all bits of counter are high
START //Controls signal to start SM (assumed sync)

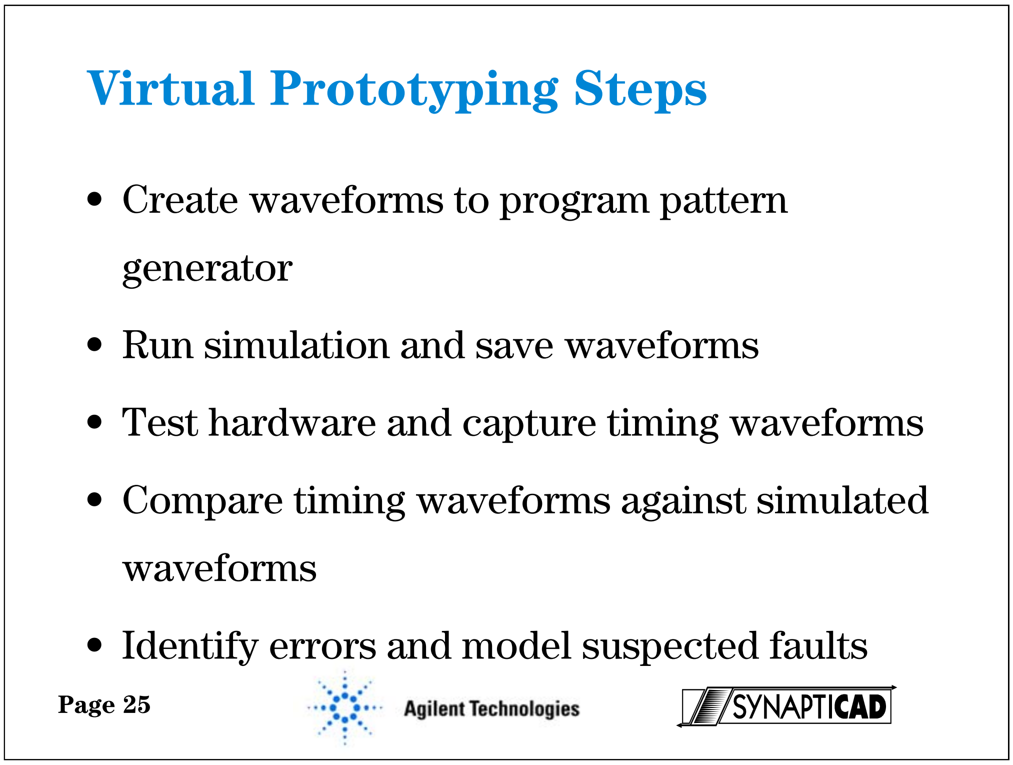
State Machine Outputs (other than states themselves)

ENABLE = READ | WRITE //Enable line for controlled devices

Note: All flip-flops are negative edge-triggered by CLK0. The notation := means clocked. The notation = means unclocked.

The equations that describe the circuit were converted to a Verilog model of the circuit for verification inside VeriLogger's simulation environment.

WaveFormer and VeriLogger can import simulation waveforms from other simulation environments, so the next several steps could also be performed using a third party simulator which can export to a waveform format such as Value Change Dump (VCD files).



Virtual Prototyping Steps

- Create waveforms to program pattern generator
- Run simulation and save waveforms
- Test hardware and capture timing waveforms
- Compare timing waveforms against simulated waveforms
- Identify errors and model suspected faults

For this case study we followed these basic steps to find out if the design would perform as expected.

● We used the timing diagram editor in VeriLogger Pro to create the ADC waveforms for the input to the physical circuit

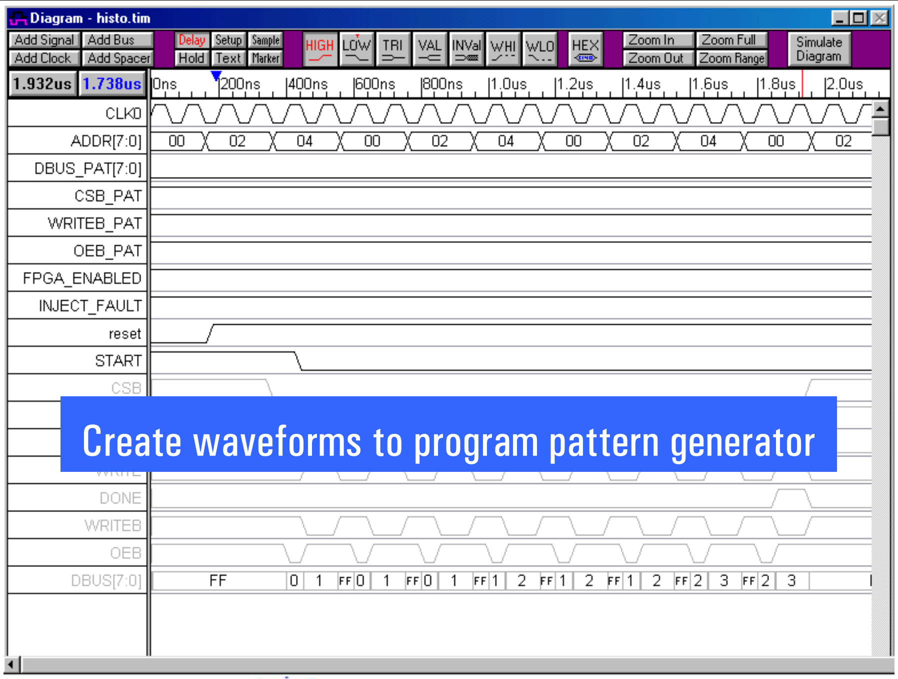
● We ran the simulation and verified that the design worked.

● Next the timing waveforms were converted to patterns for the pattern generator.

● Now the pattern generator provides input to the histogram circuit on the FPGA and the Agilent timing analyzer captures the data.

● The real timing behavior of the design is loaded into the SynaptiCAD Verilogger Pro and a comparison is made between the simulation and the real-world results. The tool automatically makes the comparisons and identifies differences.

● Let's take a look at each step



Create waveforms to program pattern generator

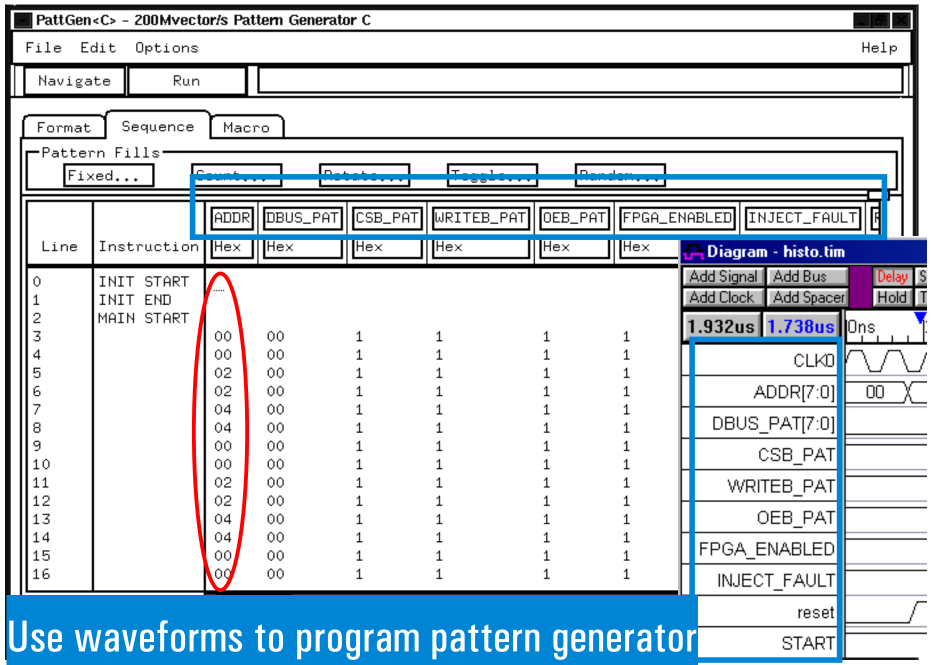
Agilent Technologies

SYNAPTICAD

●In VeriLogger Pro, load the project containing the simulation models and test benches.

● Click on the “Build Active project” icon and then click the “Run Simulation” icon. VeriLogger Pro will take the Verilog design files and simulate the design.

● Use the “Export” menu functions to export the simulation waveforms to a file that can be read by the Agilent Technologies Pattern Generator.



Line	Instruction	ADDR Hex	DBUS_PAT Hex	CSB_PAT Hex	WRITEB_PAT Hex	OEB_PAT Hex	FPGA_ENABLED Hex	INJECT_FAULT
0	INIT START							
1	INIT END							
2	MAIN START							
3		00	00	1	1	1	1	
4		00	00	1	1	1	1	
5		02	00	1	1	1	1	
6		02	00	1	1	1	1	
7		04	00	1	1	1	1	
8		04	00	1	1	1	1	
9		00	00	1	1	1	1	
10		00	00	1	1	1	1	
11		02	00	1	1	1	1	
12		02	00	1	1	1	1	
13		04	00	1	1	1	1	
14		04	00	1	1	1	1	
15		00	00	1	1	1	1	
16		00	00	1	1	1	1	

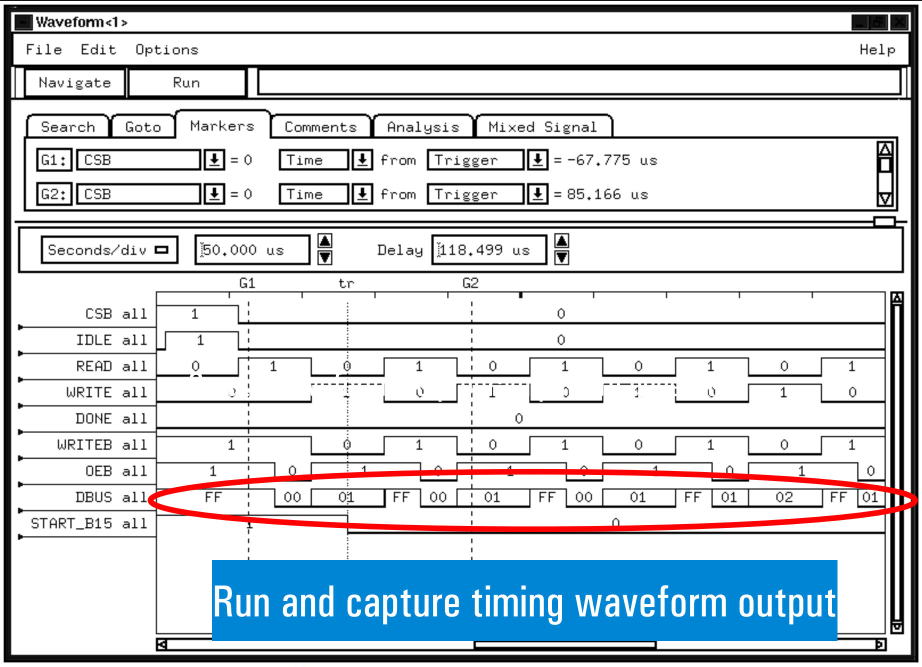
Use waveforms to program pattern generator

Agilent Technologies

SYNAPTICAD

●Transfer the stimulus file to the pattern generator. Note, that the signal names in the pattern generator automatically match those in the simulator.

●The file was transferred over the Lan. It's interesting to note that the logic analyzer can mount the C drive on the PC as its disk drive so the analyzer can just load the file from the C drive. This saves having to even to a file transfer.



G1: CSB = 0 Time from Trigger = -67.775 us

G2: CSB = 0 Time from Trigger = 85.166 us

Seconds/div 50.000 us Delay 118.499 us

```
CSB all      1  0
IDLE all     1  0
READ all     0  1  0  1  0  1  0  1  0  1
WRITE all    
DONE all     0
WRITEB all   1  0  1  0  1  0  1  0  1
OEB all      1  0  1  0  1  0  1  0  1  0
DBUS all     FF  00  01  FF  00  01  FF  00  01  FF  01  02  FF  01
START_B15 all  1  0
```

Run and capture timing waveform output

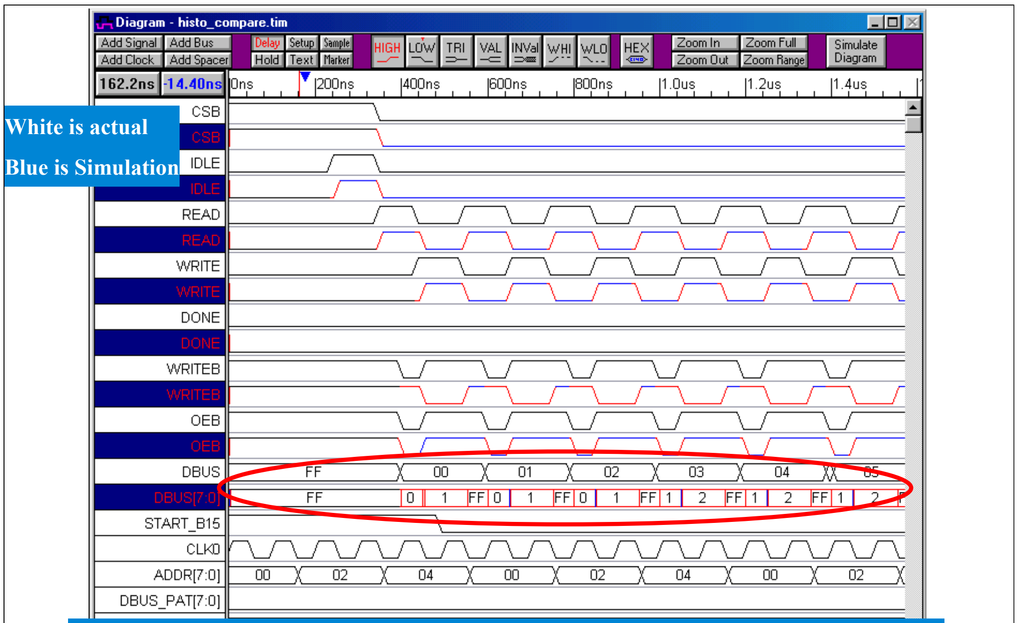
Page 28

Agilent Technologies

SYNAPTICAD

●Run the pattern generator to provide stimulus to the Xilinx Spartan XCS30 FPGA. The pattern generator is simulating a missing ADC and the missing control logic circuit.

● Use the timing analyzer in the logic analyzer to capture the output data of the histogram circuit.



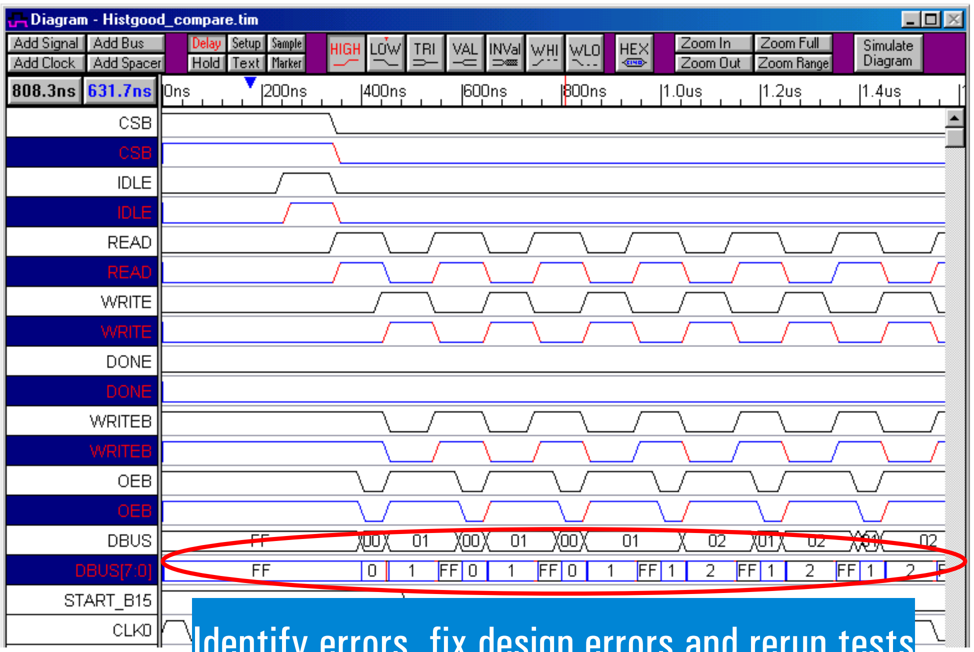
Compare timing waveforms against simulated waveforms

Agilent Technologies

SYNAPTICAD

●Use file compare in VeriLogger. This tool will interleave the simulation waveforms with the actual waveforms and then highlight in red any waveform areas that don't match.

●We can quickly see that simulation and real-time measurements don't match. There is a problem.



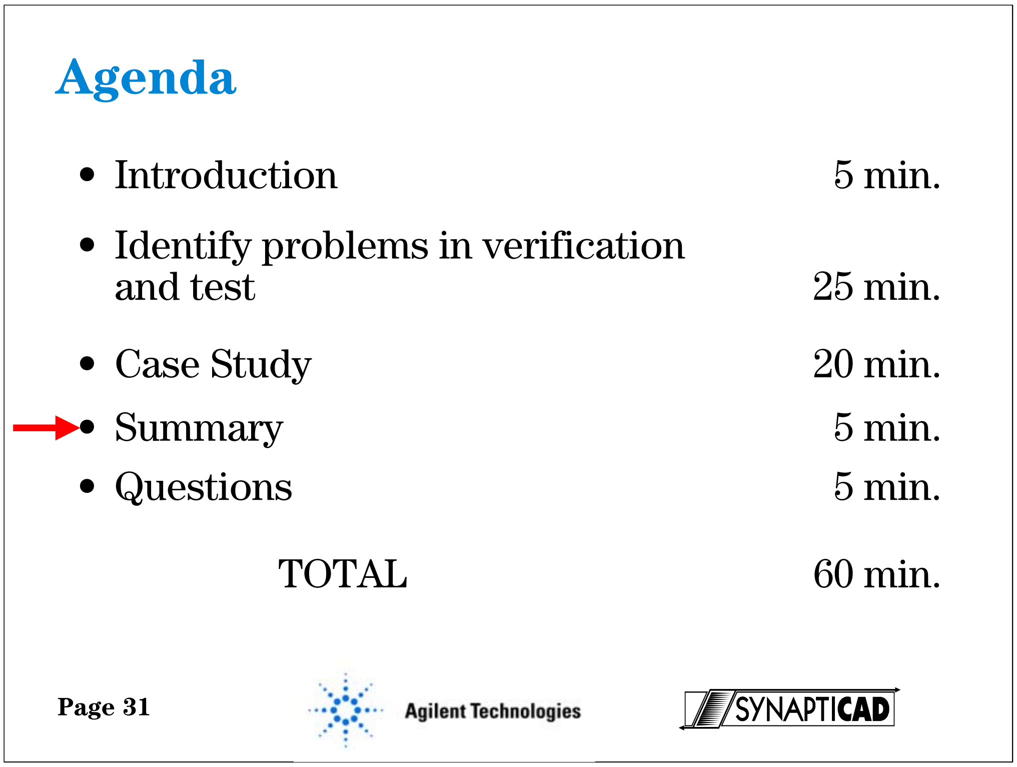
Identify errors, fix design errors and rerun tests

Agilent Technologies SYNAPTICAD

- After fixing the design error we rerun the simulation and real-time measurements and compare them again. This time the data is what we expect.

- One other interesting thing to note is if you look carefully at the data buses you notice that the simulator expected to see an FF between each change in the data. The timing analyzer did not record that in the actual circuit.
  - The reason is that the circuit is active low and the simulator was set so that the RC time constant for the pull up resistors was 0. But the actual circuit design had a fairly large pull up resistor and took some time. So the bus never had a chance to reach FF before the next clock cycle.
  - In this case this had no effect on the operation of the circuit. However, in other cases this could create major problems or worse those really nasty intermittent ones.
- By combining simulation, physical stimulus, and real-time measurements you can quickly determine if:

  1. The circuit design will work even when not all parts of the prototype board are available.
  2. That the expected simulation behavior matches the physical results.



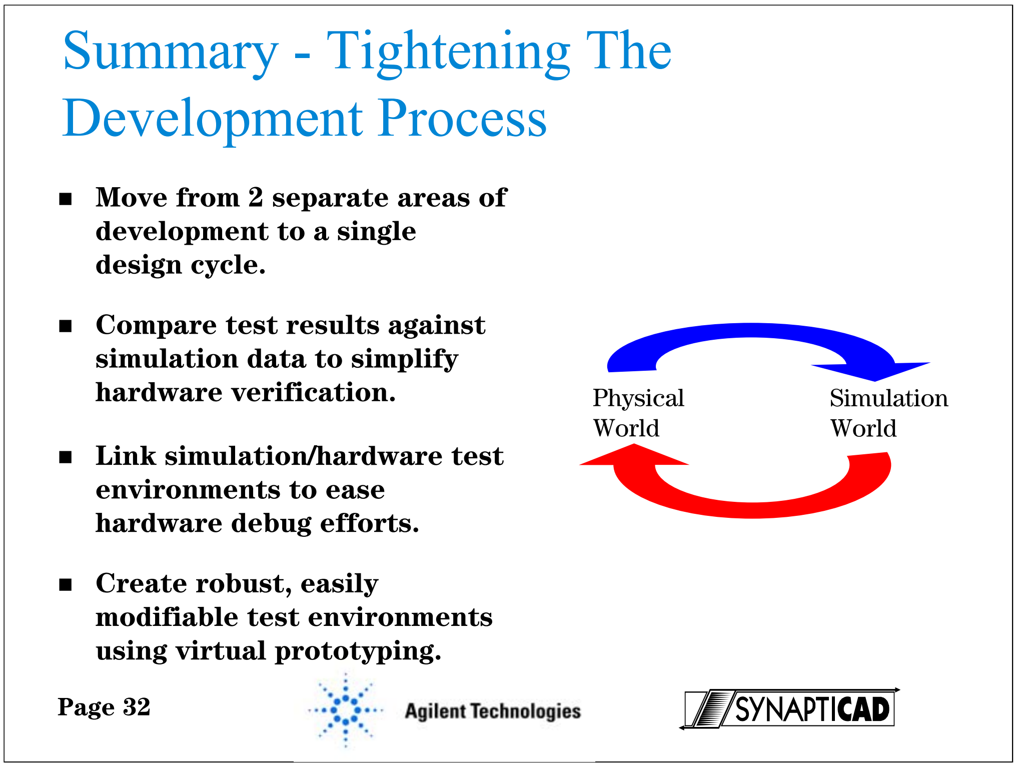
Agenda

• Introduction	5 min.
• Identify problems in verification and test	25 min.
• Case Study	20 min.
→ • Summary	5 min.
• Questions	5 min.
TOTAL	60 min.

Agilent Technologies

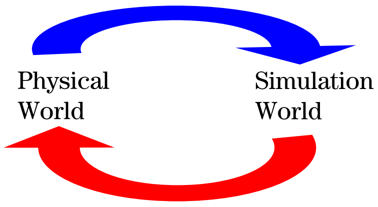
SYNAPTICAD

.



Summary - Tightening The Development Process

- Move from 2 separate areas of development to a single design cycle.
- Compare test results against simulation data to simplify hardware verification.
- Link simulation/hardware test environments to ease hardware debug efforts.
- Create robust, easily modifiable test environments using virtual prototyping.



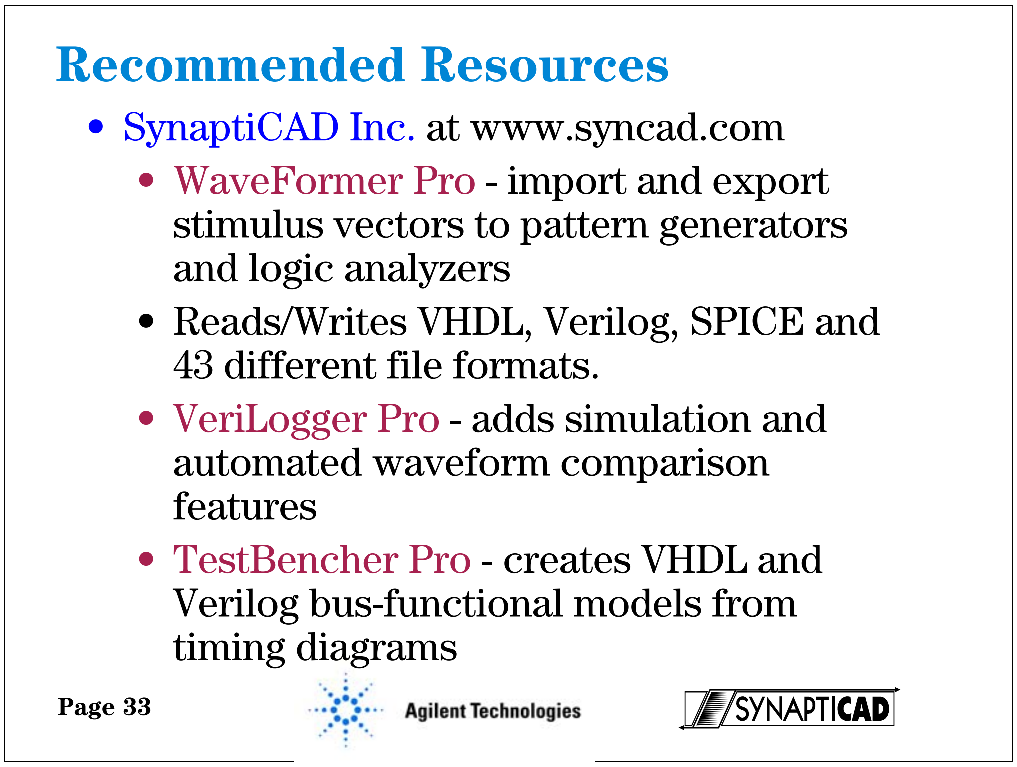
Agilent Technologies SYNAPTICAD

It is now possible to combine the power of EDA tools with test equipment to change the design cycle from four separate area of development into a single coordinated design cycle.

By bringing the worlds of simulation and hardware test closer it is possible to reduce design time and eliminate multiple hardware prototypes. No longer do projects have to be delayed when simulation alone fails to create the perfect design.

Robust test environments and even VHDL testbenches can be built quickly and efficiently using virtual prototyping. By comparing physical test results against simulation data engineers can eliminate days of tedious analysis.

By using SynaptiCAD's WaveFormer Pro tools and Agilent Technologies's logic analyzers the worlds of EDA tools and physical measurements are brought together to reduce design time, eliminate multiple prototypes and get your next design to market faster.



Recommended Resources

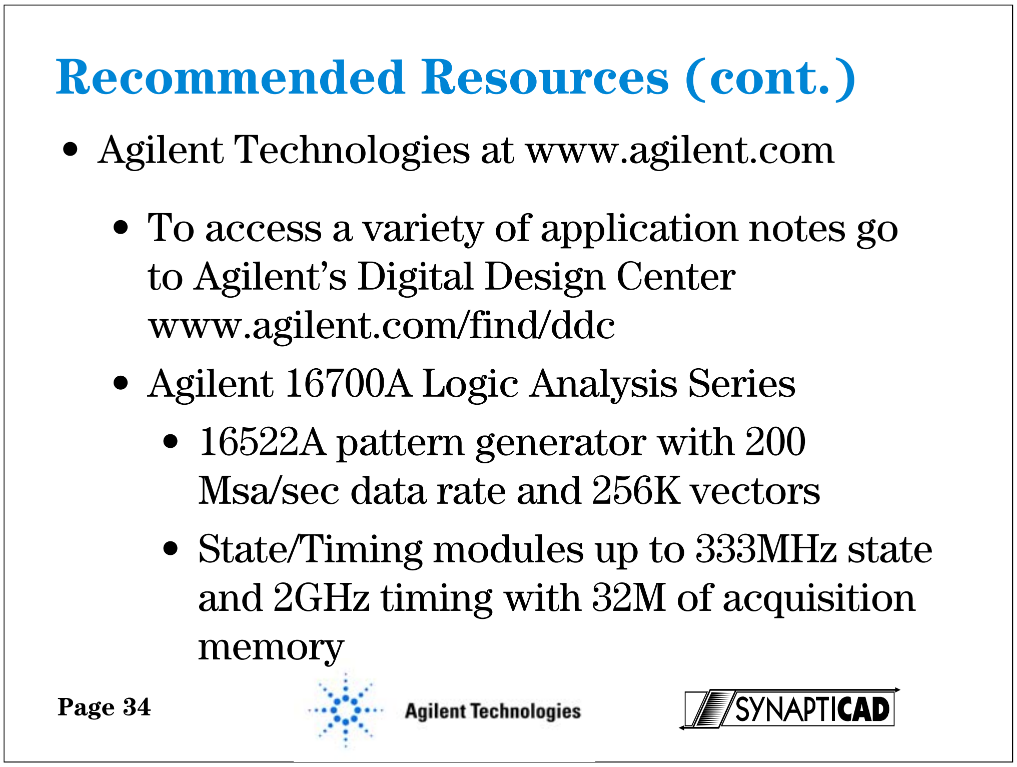
- SynaptiCAD Inc. at www.syncad.com
  - WaveFormer Pro - import and export stimulus vectors to pattern generators and logic analyzers
  - Reads/Writes VHDL, Verilog, SPICE and 43 different file formats.
  - VeriLogger Pro - adds simulation and automated waveform comparison features
  - TestBencher Pro - creates VHDL and Verilog bus-functional models from timing diagrams

Page 33

If you'll turn now to slide 33, let me run you through some of the resources we recommend for EDA assisted hardware verification. We're Synapticad, so I'd like to give you a brief description of our major products. As I mentioned earlier these are full CAD tools that can completely describe timing relationships in your digital circuits.

SynaptiCAD has three tools that support the Agilent Technologies Virtual Prototyping features. WaveFormer Pro can import and export stimulus vectors to Agilent Technologies Pattern Generators, Agilent Technologies Logic Analyzers, VHDL, Verilog, SPICE, and 43 different formats. VeriLogger Pro contains the import/export features of WaveFormer in addition to the automated waveform comparison features. TestBencher Pro in turn contains all of the features of WaveFormer and VeriLogger in addition to being able to create VHDL and Verilog Bus-functional models from timing diagrams.

Let me now turn it over to Gregg, who'll describe some of the Agilent resources.



Recommended Resources (cont.)

- Agilent Technologies at www.agilent.com
  - To access a variety of application notes go to Agilent’s Digital Design Center www.agilent.com/find/ddc
  - Agilent 16700A Logic Analysis Series
    - 16522A pattern generator with 200 Msa/sec data rate and 256K vectors
    - State/Timing modules up to 333MHz state and 2GHz timing with 32M of acquisition memory

SynaptiCAD has three tools that support the Agilent Technologies Virtual Prototyping features. WaveFormer Pro can import and export stimulus vectors to Agilent Technologies Pattern Generators, Agilent Technologies Logic Analyzers, VHDL, Verilog, SPICE, and 43 different formats. VeriLogger Pro contains the import/export features of WaveFormer in addition to the automated waveform comparison features. TestBencher Pro in turn contains all of the features of WaveFormer and VeriLogger in addition to being able to create VHDL and Verilog Bus-functional models from timing diagrams.

Agilent Technologies make a variety of Pattern Generators and Logic Analyzers. SynaptiCAD’s tools supports almost the entire line of equipment.

Question and Answer Time

Don't forget to fill out your survey to enter in the drawing for a Palm Pilot.

Agilent Technologies
Innovating the HP Way

SYNAPTICAD