By creating appropriate digital data, you can test individual components before the entire system is ready.

Shawn Beus, Hewlett-Packard

Separately Testing Subsystems before the Entire System Exists

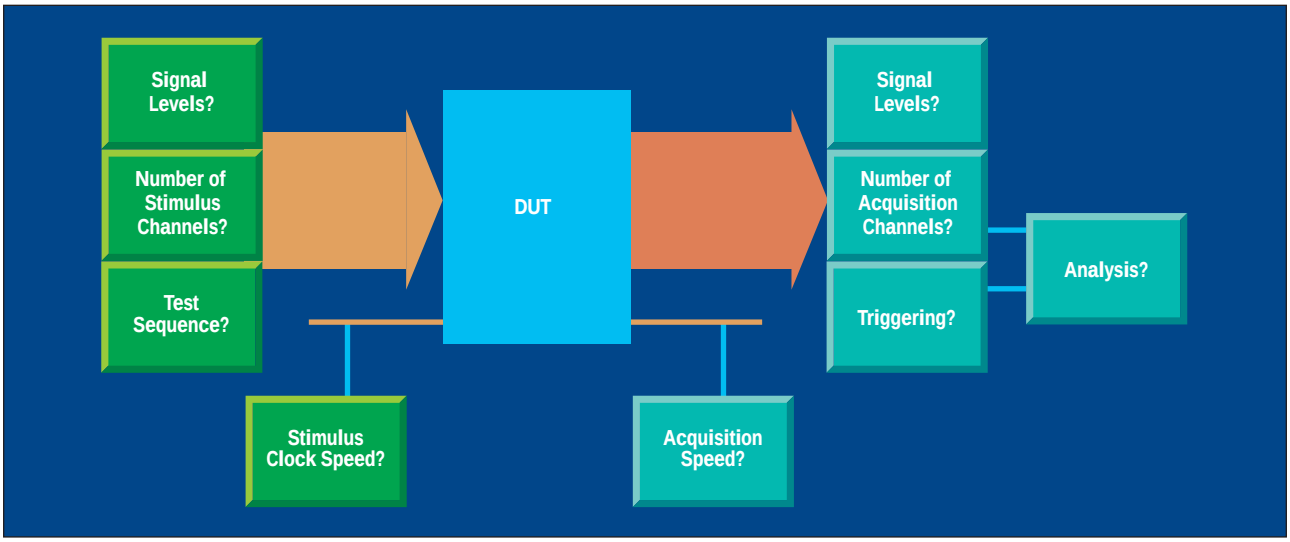


Figure 1. To functionally test a device, you must know what signals the rest of the system provides and the response the device is supposed to produce.

Functional testing of components and subsystems is an integral part of the development process. If you begin testing when only part of the system is ready, you can uncover and solve problems earlier and be further along in the debugging process when all of the system hardware finally comes together. Once you have a functioning digital system, you can stimulate it with various data sequences to determine whether it is performing as designed under as many conditions as possible. You might want to simulate data from I/O devices to see if the system processes it correctly, or introduce faulty sequences to see how it performs under stress conditions. For any type of functional test, you need tools to input digital stimuli into the system under test and to measure its response. Using the right tools in the right way will speed the development process and result in a more reliable system.

Before you set up a functional test, you need to assess the test requirements for your system and then choose the appropriate tools (**Figure 1**). First, you must determine what basic test capabilities are needed, such as the number of stimulus channels, the required stimulus clock speed, the voltage standard of the input, and the length of the test sequence. You may test the system at a single frequency or at different frequencies. The test sequence may simulate a single serial data stream or it may stimulate several buses simultaneously.

Another important consideration is how to view and/or measure the system's response to the test sequence. For simple tests, you can design the test to output its results to a monitor or an output port. If you are testing a single chip such as a digital-to-analog converter, you can connect an oscilloscope to the output.

However, to see the full response of a digital system to test sequences, you have to look simultaneously at a number of data streams in the system, and a logic analyzer state/timing machine is the most appropriate tool for this. A logic analyzer also can help you trace the events leading to an error condition.

Complex Tests Require Flexible Tools

Other considerations are the amount of flexibility needed to perform the tests and the time and

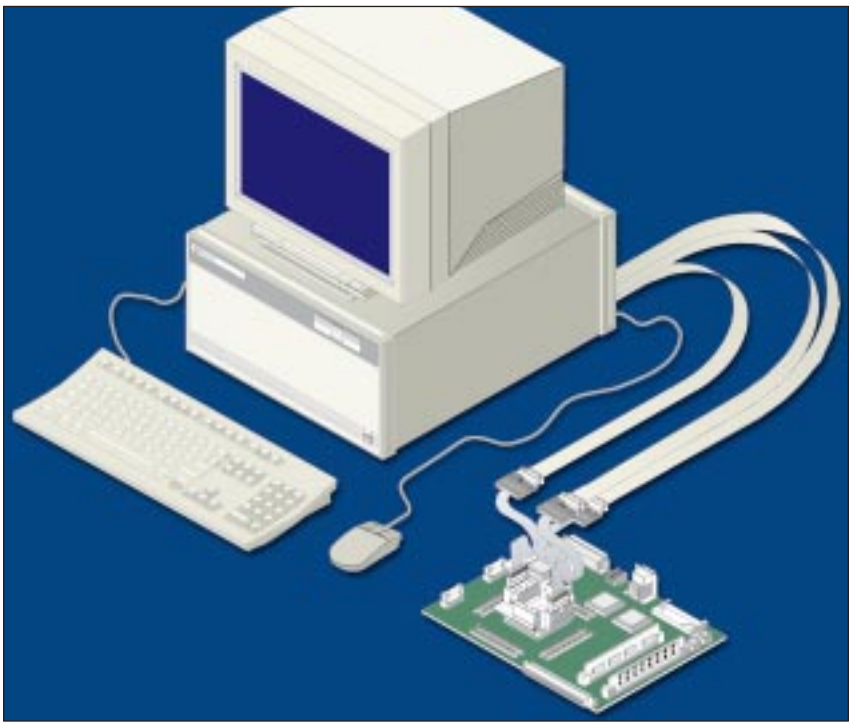


Figure 2. Connecting many signals to or from the DUT is often difficult unless you either plan ahead by placing some connectors on the DUT or use a special attachment device.

budget constraints. If you have only one simple test sequence to perform, you may consider developing your own state machine to provide the test stimulus. This approach can take anywhere from a week to several months, depending on the complexity of the state machine. The largest benefit of this method is the low cost of the materials, but this low cost is often offset by the expense of development. This approach results in a stimulus tool suited only for a single test. Any modifications to this custom tool would require additional development time, and a new stimulus tool would need to be developed for each new project. If the required test sequence is complex or if a flexible stimulus tool is required, the true cost of a custom state machine would be astronomical.

Fortunately, there is a solution to this dilemma. A general-purpose pattern generator provides more flexibility for testing a digital circuit. The initial cost of a pattern generator may be higher than the cost of developing a state machine, but the savings in development time compensates for this cost for all but the simplest tests. A few hours are required to become familiar with the pattern generator, but many hours are saved because the tool allows test sequences to be easily developed and modified. Furthermore, once purchased and learned, it can be used over and over again on new projects with no additional cost.

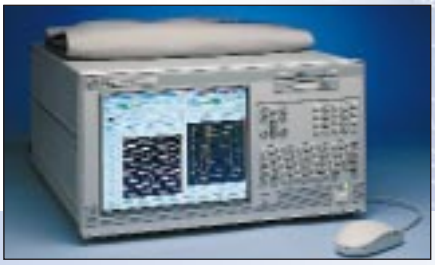
After selecting appropriate tools, you need to determine how to connect them to your system. Early in the design process, you can design connectors into the system specifically for use by the stimulus tool. This approach greatly reduces the chance of misconnecting a signal or having signals become disconnected during testing. If this is not possible, the stimulus tool can be connected through various pins and connectors that are already part of the system.

If a logic analyzer will be used to capture the system's response to the test sequences, you also need to plan for connecting it to your system. If the system contains a microprocessor, analysis probes are available for many standard microprocessor families. Analysis probes often include inverse assembly software that decodes the address, data, and status lines into assembly language for easy understanding of the response. If you need to capture bus activity to evaluate the response, analysis probes are also available for standard buses. If the system contains a proprietary bus or chip, a connector or general-purpose probe is required to access the necessary signals (**Figure 2**).

Setting Up the Test Sequence

The next step in planning a functional test is to determine how to set up the required test sequence in the pattern generator. The user interface of the pattern generator normally allows you either to generate standard test sequences such as "walking 1s" or random numbers, or to enter a test sequence one line at a time. Writing and entering test sequence data by hand is time-consuming and tedious, so it is preferable to automate this process.

Simulation tools, which are used to test a circuit design before actual hardware testing, are one



Create Stimulus and Analyze Response in a Single Instrument

The HP 16700A logic analysis system and the HP 1660EP series of benchtop logic analyzers offer stimulus/response functionality in a single instrument. The HP 16700A, which can be configured with both pattern generator and logic analysis cards, provides one integrated system for applying stimulus to the system under test and for measuring the system's response. The pattern generator can output a test sequence at speeds up to 100 MHz across 200 channels or at speeds up to 200 MHz across 100 channels. It supports ECL, TTL, 3.3-V, and CMOS voltage levels and has 256 Kb of memory per channel to hold test data. The pattern generator outputs can also be set to tri-state if the input channels need to be driven by other devices. The various state/timing cards for the HP 16700A allow you to capture the response of your digital system across 340 channels. And with trace memory depths from 8 Kb to 2 Mb per channel, you can choose the length of the response trace to capture.

The HP 1660EP series of benchtop logic analyzers offer similar functionality in set configurations at lower cost. This series provides an integrated pattern generator that can output a test sequence at speeds up to 100 MHz across 34 channels or at speeds up to 200 MHz across 16 channels. For example, the HP 1663EP has a 32-channel pattern generator for stimulus and a 32-channel state/timing machine for capturing system response.

For additional information on products mentioned in this article, check 3 on the reply card, or visit http://www.hp.com/info/insight3.

source of test sequences. Most EDA tools allow routing test sequences to an ASCII file at various points in the circuit. In the EDA tool, you can save the test sequence at the point where the pattern generator is connected to the system, add some setup information at the beginning of the file, and then load this sequence into the pattern generator.

To simulate a working system component with the pattern generator, you can capture the output of this component with a logic analyzer state machine and load the captured sequence directly into the pattern generator. You can modify this test sequence manually to create variations on the original sequence. Variations that introduce a few errors are useful to show how the rest of the system responds to adverse conditions. If the component being simulated isn't working properly, you can correct its output before loading it into the pattern generator and continue developing the rest of the system while the faulty part is being fixed.

Another method for obtaining a test sequence is to write a simple computer program to create an ASCII file that can be loaded into the pattern generator. A program can be modified easily and produces long test sequences much faster than they can be entered by hand.

Synchronizing the Clock

Once the test sequence has been created and entered into the pattern generator, then synchronization and clocking must be considered. For certain applications you may need to synchronize the start of the test sequence with events occurring in the system under test. For example, you may want to simulate an I/O device with a pattern generator and need to start the test sequence when the system is ready to read from the device. In this case, you can connect the read signal to the wait inputs of the pattern generator and have the pattern generator look for a read before starting the test sequence. If you want the pattern generator to respond to a series of system reads, you can set up multiple data sequences with a wait before each one. The pattern generator will respond to each read with the corresponding data sequence and then wait for the next read.

If the test must wait on system events too complex for the pattern generator to recognize, you can set up a logic analyzer to trigger on the event that signals the start of the test sequence. To simulate a memory-mapped I/O device, set up the logic analyzer to trigger on a read from that device's address and then have it start the pattern generator.

Clocking is another issue in

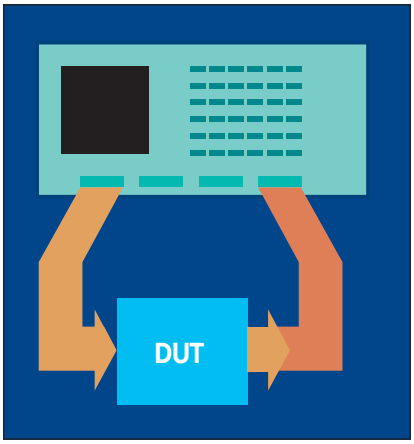


Figure 3. Instruments that combine pattern generation with logic analysis ease the process of functional testing.

designing a functional test. Several options are possible: the clock in the pattern generator, the clock from the system under test, and an external signal generator. The pattern generator's internal clock provides an easy way to clock the whole target system. It can be set to a variety of periods and can be delayed with respect to the data. For example, you may want to verify that the lines to the memory subsystem are routed correctly even when the system processor is not yet available. You can set up the pattern generator to write data to the memory while using its internal clock to run the system. Then you can use the pattern generator to send control signals to read the same values back from memory while a logic analyzer captures the values. If the corresponding memory values don't match, the stimulus and response data can be compared to identify which board traces are faulty.

For some test situations you may want to use a clock external to the pattern generator, either because you need more precise control over the clock or because you want to use the clock of the system under test. The pattern generator will run on an external clock, but there is some time delay from clock in to data out. This delay is due to the propagation time as the clock signal travels up the pattern generator cables and the data travels back down. If you are using a signal generator to provide the clock signal, you can use the delayed version of the pattern generator clock to clock the test sequence into your system. If you are using the system clock to drive the test sequence, you need to take into account the propagation delay from clock in to data out. You may want to use a logic analyzer timing machine to verify that the pattern generator outputs are timed correctly with respect to the system clock edges.

Many scenarios are possible using a pattern generator to run a test sequence while a logic analyzer state machine captures the response (**Figure 3**). If you just received first silicon of a custom ASIC and want to verify its functionality, you can set up the pattern generator to drive the input lines to the ASIC while a logic analyzer captures the response on the output lines.

In another case, you can simulate a video image to test whether a video processing system handles it correctly. You can load the pattern generator with the image and set up the logic analyzer to trigger immediately to capture the processed image. This test could be repeated multiple times and the various responses compared to determine how much variation there is in the image processing system's output.

Repeat Specific Sequences to Track Down Intermittent Errors

Sometimes, an anomaly shows up intermittently when a test sequence is performed. In this case, a pattern generator can be set up to repeat a test sequence continuously while a logic analyzer is set up to trigger on the error condition. For intermittent errors in a digital communication system, you can set up the pattern generator to continuously supply test data to the transmitting device and set up the logic analyzer to monitor for errors at the output of the receiving device. If the logic analyzer detects an error, it will trigger and display a trace of the faulty transmission. The trace data can be evaluated to find the root cause of the problem.

The ability to functionally test a digital circuit at all development stages makes the process go more smoothly. By testing custom parts separately or simulating missing or faulty parts, you can debug the available parts of your system before the whole system is complete. When all of the system hardware comes together, you will have already fixed many of the problems and will be able to deliver a working system in a shorter time. Finally, by stimulating a working system with a variety of test sequences and fault conditions, you can better verify that it functions correctly. Functional testing can help you meet product schedules and beat time-to-market pressures.

For additional information on products mentioned in this article, check 3 on the reply card, or visit http://www.hp.com/info/insight3.