

# Speeding SOC test

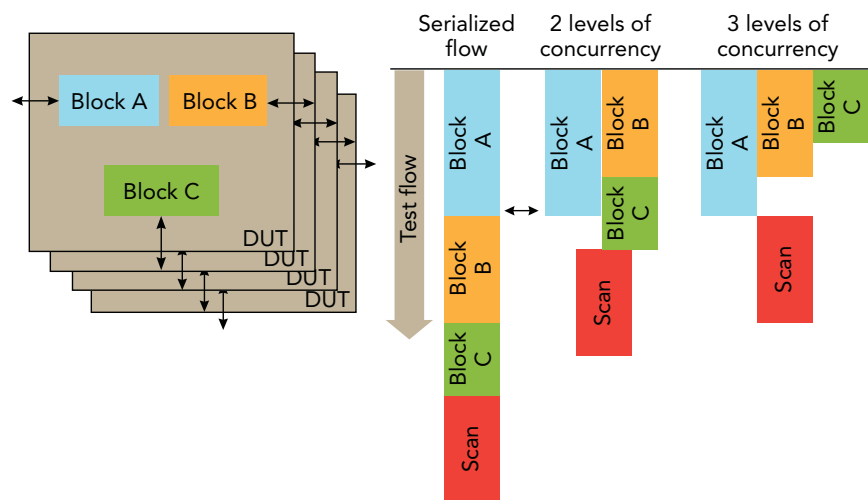
Protocol-aware ATE hardware boosts the efficiency of parallel-site and concurrent tests for SIP, SOC, and MCM designs.

BY TIM LYONS, TERADYNE

Integrated semiconductor products often include building blocks from a variety of sources that are implemented as an ASIC, as individual dies on a common substrate, or as stacked dies connected through vias. In all cases, individual blocks commonly have their own time base, communication bus, and perhaps even free-running clocks. For example, the processing unit of a netbook or tablet may have interfaces that let it communicate with external networks, high-definition televisions, and cellular networks as well as interfaces for internal PCI, memory, flash, and display buses.

Each of these interfaces has a different standardized timing and protocol, and testing each of them with a traditional tester can greatly slow down your throughput (**Figure 1**). A better option for production test of complex packages is to perform multisite parallel testing with an ATE (automated test equipment) system that includes a PA (protocol-aware) tester.

For test purposes, each separate block of a semiconductor requires a digital test instrument to align itself in frequency, phase, and data against the output timing of that block. Although the exact frequencies are known in the test environment, because they are mathematically derived from tester-sourced input clocks, the phase timing of output edges and the location of data words within the larger stream of bits are unknown. For traditional ATE architectures, the test pro-



**FIGURE 1.** Complex devices implemented with concurrent blocks present test-management and programming challenges for both concurrent test and parallel-site test.

grammer must implement a match loop using a common fail flag to search for the timing of the output edges and must implement a search for unique patterns buried in continuous data streams. The programmer also must manually compensate for round-trip delay, digital pipelines, analog latency, and asynchronous time bases. The **table** shows the typical logical programming flow for testing interactions among concurrent blocks in a device. The flow is repeated serially for each test site.

### Verifying functional performance

The complexity of programming and test-instrumentation management can make it difficult for engineers to verify a

device's functional performance and the interactions among its separate blocks. A traditional tester typically has one or two fail flags that must be shared among blocks and must be reset by the test controller between each timing search. The resulting measured and corrected timing for a block will need to be reprogrammed with a slow asynchronous test controller broadcasting new timing to all of the affected instruments. The programming process is handed off for the next block and repeated serially.

Parallel-site testing further complicates the testing of asynchronous blocks. After a single site is programmed correctly, the process is handed off and re-

peated for other parallel test sites and repeated again for every timing, level, and temperature parameter. The problem rapidly becomes a daunting programming challenge that complicates code maintainability and leads to inefficient testing. For example, every 1% of serialized overhead in single-site test will translate into an 8% reduction of parallel efficiency for octal-site testing, assuming all else is 100% parallel efficient. Clearly, improvements in efficiency have real benefits in reducing test costs and improving capital equipment usage.

You can resolve these test challenges by using an ATE system that includes hardware-based PA test. A PA tester can stimulate the device under test, measure and interpret the device response, and then re-stimulate the device with new data based on the timing and message content of the device response, all without the need for a programmer to write code. The PA tester can form the stimulus and response in compliance with standard protocols that define variability in bit patterns, data frequency, error encoding, latency of response, and code scrambling.

**A USB 3.0 example**

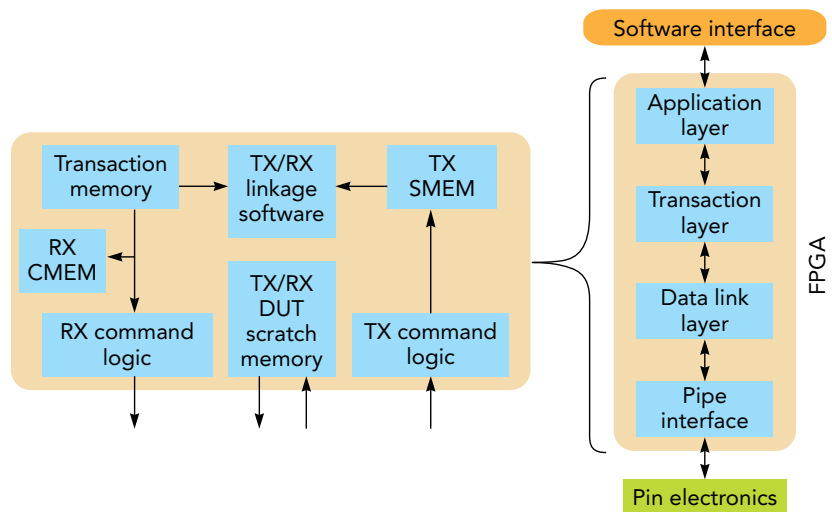
For example, a USB 3.0 interface may be one of many bus interfaces incorporated in a larger integrated device. For tests performed with a traditional tester architecture, a programmer must derive the cor-

rect packetized data from the USB 3.0 standard, perform 8b/10b encoding on the data, and program the tester to stimulate the interface at a nominal 5 Gbps and be compliant to the wake-up sequence.

After a varying time allowed by the standard, the device provides a response to the wake-up sequence at a nominal data rate, which may be slightly offset or slightly modulated with spread-spectrum clocking. The programmer must implement a match loop to derive the phase and frequency of the output bits, reprogram the digital-receiver timing, and then implement a second match loop to detect the packet boundaries. There may be a need for a third match loop to locate the actual packet data.

After all these interactions, if the wake-up sequence is just part of the larger wake-up of the whole device, the programmer must perform match loops to wake up the other buses on the device. Then, the process must be repeated for the next parallel site. Due to the limits of traditional testers, each step uses one of only a few fail flags and relies on a central test controller to reprogram timing.

To improve this process, Teradyne has developed a PA engine that provides a new programming interface (Figure 2) for its UltraFLEX tester. Local to the digital pin electronics, the hardware-based PA engine can stimulate and respond to the DUT (device under test) in compliance



**FIGURE 2.** The UltraSerial10G protocol-aware test instrument for Teradyne’s Ultra-FLEX platform includes parallel independently programmable protocol engines with local intelligence for measuring device response and providing test stimulus. The instrument includes logic and memory blocks, including RX CMEM (receive capture memory) and TX SMEM (transmit source memory).

## Table. Logical programming flow in traditional ATE architectures

### Start test:

Power and provide input clock to block A  
Power and provide input clock to block B  
Power and provide input clock to block C

### Block A:

Ignore blocks B and C  
Program nominal output timing for block A  
Assign shared time measurement to block A pins  
Enter search loop to detect block A output timing  
Reprogram block A output timing

### Block B:

Ignore blocks A and C  
Program nominal output timing for block B  
Assign shared time measurement to block B pins  
Enter search loop to detect block B output timing  
Reprogram block B output timing

### (Repeat for all logic blocks)

Functionally stimulate all blocks  
Functionally test all blocks' outputs

with the selected protocol. For the USB 3.0 example, the local engine initiates the wake-up sequence and expects a compliant response. It can locally decode and encode packets and data, and it can control timing in compliance with the standard. Only for the highest data rates will the local decoding not keep up with received packets and require non-real-time processing. The protocol engine detects DUT timing and measures the timing values. If the values are not in compliance with the standard, the tester reports the result to the test controller, and the DUT is failed or retested.

The test programmer can focus on understanding the data and messaging and is relieved of the burden of force-fitting an inherently statically predicted digital system to the variable timing and encoding of a semiconductor device's complex protocol. The complex series of steps described in the table could be replaced by a simple series of calls already verified to be compliant with the relevant standard, such as:

```
“dataOut = pcie(write, Addr, <array>)”
```

Although conceived as an efficient method for rapidly developing tests for standard interfaces, the addition of a hardware-based PA tester enables an ATE system to rapidly detect an output clock, align itself to the clock's phase, and correctly detect and decode data words within the related interface bus. This functionality also has the potential to greatly improve throughput.

For example, a complex test program can have well over 100 pattern starts. If each pattern start requires a phase alignment against the DUT output data, there

will be 100 match loops per interface that may need to be serialized per interface. Each match loop needs to be followed with a timing reprogram for the digital instrument connected to the interface.

For a traditional tester, each timing reprogram and fail flag reset may consume 5 ms of tester time. It's not unusual to spend 500 ms of total time that then serializes across multiple parallel test sites. A built-in PA tester can achieve much faster round-trip performance. The 5-ms match-loop and reprogram time can be reduced to as little as 100 ns, and the programming code will be easier to maintain.

In fact, a PA test architecture can provide independent hardware PA engines to every eight digital pins, providing true parallel-test capability across the entire DUT for all test sites. The negative impact on parallel test efficiency caused by match loops vanishes.

A full PA hardware architecture not only speeds test development and improves programming code maintainability, it also has the unexpected side benefit of providing improved test capability for parallel and concurrent test, as it distributes parallel control and response across the digital instruments. As a final bonus, PA directly addresses the thorniest of parallel test-programming challenges. **T&MW**

**Tim Lyons** is an applications engineer in the Semiconductor Test Division of Teradyne in North Reading, MA, with 24 years of experience. He is focused on SerDes, jitter, and high-frequency analog test solutions. Lyons has an MSEE degree concentrating in digital-signal processing from Tufts University.