LAWRENCE VIVOLO | Synopsys Inc.
lvivolo@synopsys.com

Transaction-Based Verification And Emulation Combine

For Multi-Megahertz Verification Performance

Transaction-based verification is becoming the emulation mode of choice because of its unique ability to keep pace with the growing verification support requirements of the latest high-performance, industry-standard interfaces.

Design complexity has grown with each successive generation of system-on-chip (SoC) evolution. SoCs now include many industry-standard interfaces, multiple processor cores, on-board memories, and embedded firmware and software. With this growth, we've seen verification complexity grow exponentially, driving engineers to seek advanced verification methodologies.

While in-circuit emulation (ICE) promises megahertz verification performance, its dependence on protocol-specific rate adapters combined with the sheer number of high-speed, protocol-specific physical interfaces has proven to be a recent challenge for today's advanced designs. Transaction-based verification (TBV) holds the key to bridging the performance of emulation with the flexibility and simplicity of simulation.

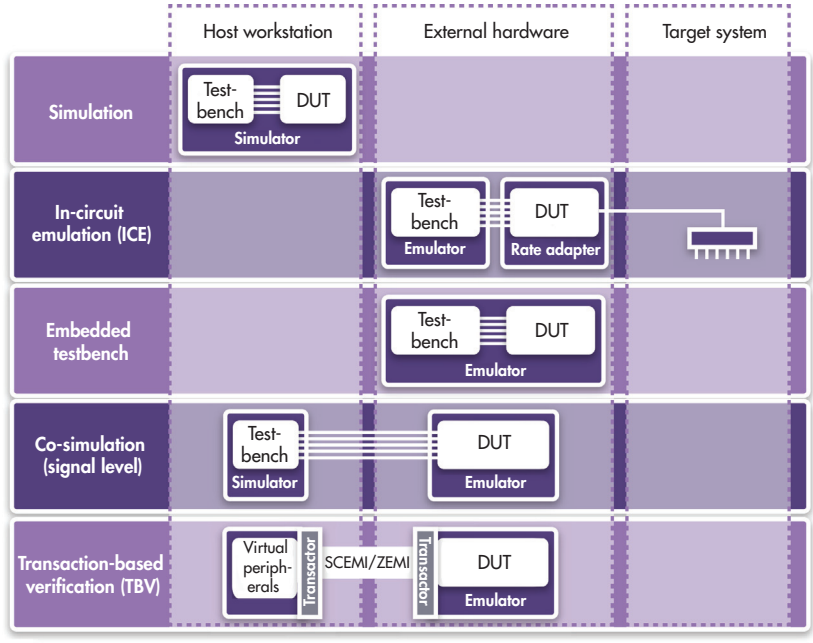
By adopting accelerated transaction-based emulation strategies, designers can move their verification strategy up a level of abstraction and achieve the leap forward in verification performance and productivity that is necessary to fully debug and develop the most complex electronic hardware and software-based systems. Different use modes are available in today's emulation systems, and TBV is becoming the preferred emulation mode for achieving multi-megahertz verification performance.

SIMULATION

Today, simulation continues to be the workhorse of complex verification. Most verification is done using simulation, finding more than 95% of bugs. Simulation usually runs on a single workstation, and its environment consists of two components: the design under test (DUT) and the stimuli generator—the testbench *(Fig. 1)*.

With simulation, the design runs at a wire level and is clock accurate, which means there is a wire-level interface between the testbench and the DUT. For a full chip, these wires may be the device pins. With the addition of various monitors and checkers, the interface between the DUT and testbench can grow to include thousands of wires.

Simulation is still what's used for most register transfer level (RTL) development and design. Nevertheless, for designs requiring hardware/software co-verification, as well as those



1. This graphical description of the most commonly used emulation modes highlights the location of the testbench, the design under test, any rate adapters and physical interfaces, and potential performance bottlenecks.

utilizing many industry-standard protocol interfaces, maximum verification performance is required. Simulation performance is often limited by the performance of the host PC or workstation on which it executes. A large SoC, designed to run at 1 GHz, may clock at 1 Hz or less in simulation on a standard PC.

To put that into perspective, if the operating-system (OS) boot-up sequence for a cell phone, whose processor is running at 1 GHz, were to take 10 seconds, the same boot-up sequence, simulated on a host PC at 1 Hz, would require 1 billion seconds (317 years). Very few design teams enjoy a project schedule that can accommodate that.

As a result, more and more design teams are adopting some form of hardware-assisted verification. Today's high-performance emulators run very fast and support many operating modes. Synopsys' Zebu, for example, reduces the boot-up time for the above example from more than 300 years to less than an hour with its transaction-based verification mode.

IN-CIRCUIT EMULATION

The first widely used emulation platforms were introduced in the early 1990s and were limited to ICE. With ICE, the idea is to eliminate the host PC or workstation, which was (and still is) the performance bottleneck. First, the DUT is moved from the host PC to dedicated emulation hardware. Next, a live "target system," or a physical device, replaces the PC-based testbench.

A physical cable connects the two. The cable is often plugged into the very socket that the SoC will eventually plug into. This combination eliminates the host workstation or PC, allowing the design to run several orders of magnitude faster, typically at the maximum performance of the emulator itself.

ICE is not without its challenges, however, as an emulator is rarely fast enough to run at the same speed at which a target system would normally operate. In the early days, the entire target system had to be slowed down to match the speed of the emulator. This tended to be very complicated, and many devices, such as disk drives, simply could not be slowed down. Thus, rate adapters were introduced.

A rate adapter acts as a buffer, collecting all the required information from the DUT and—once the data (wire-level signal changes) is sufficiently cached—sending it at real-time speed to the target system. It also captures data from the target system at full speed, caches it, and then plays it back at emulation speed to the DUT.

While this approach works well for fairly simple protocols, today's SoCs include many complex, high-speed interfaces, each of which would require a rate adapter to be connected to an emulator. Even when a rate adapter is available, the constant evolution of speed and complexity of individual protocols (such as SATA2 versus SATA3) has made timely rate adapter development difficult. Speed requirements alone typically delay, and in more recent cases eliminate, the availability of rate adapters for the latest protocols.

Another characteristic of ICE is the simple fact that the physical devices connected to the emulator often require an operator to be physically present—for example, to push a reset switch or power-cycle the target system. This makes it difficult to leverage an emulator that is centrally located.

EMBEDDED TESTBENCH

In an attempt to leverage the existing simulation environment, as well as to avoid some of the limitations of ICE, many design teams in the mid-1990s adopted an embedded testbench approach that moves the entire simulation testbench onto the emulator. This eliminates the physical interfaces required by ICE, and with them, the need for rate adapters.

Unfortunately, such a testbench must be synthesizable to be loaded onto an emulator. Because virtually no simulation testbenches are synthesizable in today's advanced verification environments, this method is limited in use.

A variation (special case) of this approach is commonly used today. Leveraging the emulator's large reserves of physical memory, designers load this memory with system software, such as firmware, drivers, or an entire OS. The DUT's embedded processor can then run that software directly as if it were running in the actual system, effectively making the software a testbench. This approach is relatively simple to implement, and since there are no physical connections, the emulator runs at its highest level of performance

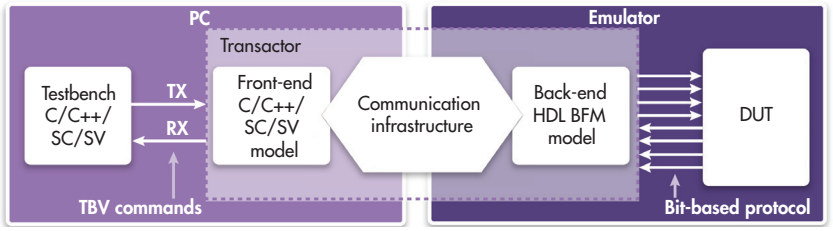
As the software is only being run through the processor, this approach still lacks the ability to test interfaces between different chip protocols. Additionally, because it requires a fairly complete processor implementation, it can only be used relatively late in the design cycle. As a result, it is generally best suited for late system-level hardware verification and early hardware/software bring-up.

CO-SIMULATION

With co-simulation, also called simulation acceleration, the DUT is moved to the high-performance emulator as with the other use models. The testbench, though, remains on the host PC or workstation. A wire-level interface connects the two.

This verification approach is very straightforward, as it leverages the existing simulation testbench and eliminates the need for external rate adapters. However, the actual verification performance is heavily limited by the performance of the host PC, the complexity of the existing testbench, and/or the wire-level interface between the testbench and the DUT.

In a typical verification environment, there are thousands of signals at the interface, with many changing multiple times within each clock period. With wire-level co-simulation, each signal must be transferred between the testbench and DUT on



2. In the transaction-based verification environment, the back-end portion of the transactor and the DUT, which remains unchanged, are located within the emulator. This requires that they both be written with synthesizable RTL.

every transition. Even though the emulator can run orders of magnitude faster, it must wait for these transfers to complete.

In addition, the emulator will often be stalled by the performance of the testbench itself. If tightly coupled to the testbench, which is common in pure simulation environments, the emulator will quickly stall, waiting for the slower testbench to react to incoming signals and produce the next set of stimuli. As a result, most designers do not use this mode for general verification. But they do use it as a stepping stone (to validate the DUT functionality on the emulator) for TBV.

TRANSACTION-BASED VERIFICATION

TBV raises the level of verification abstraction away from a wire-level interface to run thousands to a million times faster than simulation on a host PC. It simplifies the communication between the testbench and DUT so a design team can access the full performance benefit of the emulator.

With TBV, like co-simulation, the DUT is loaded onto the emulator, and the testbench resides on a host PC. But instead of a wire-level interface, TBV uses a high-performance interface using transaction-level protocols that are executed on the emulator. This transaction-level interface allows the testbench to be tightly or loosely coupled to the DUT.

The communication between the testbench, now working at a protocol level, and the DUT, which still requires a wire-level interface, is accomplished through a transactor. The transactor converts the high-level commands from the testbench into the wire-level, protocol-specific sequences required by the DUT, and vice versa. The key is that all the wire-level communications are wholly contained within the emulator itself and run orders of magnitude faster as a result *(Fig. 2)*.

Another key difference of transaction-based verification is that the transaction-level interface allows the testbench to stream data to the DUT, which the transactor buffers automatically. This speeds up the execution of the testbench. With this methodology, it is also possible to have multiple transactions active across multiple transactors. Together, these transactors enable the emulator to process data continuously, which dramatically increases overall performance to that of a pure ICE environment. As noted earlier, this is orders of magnitude faster than any wire- or pin-based verification approach.

Beyond performance, TBV offers designers several other advantages. Primarily, it eliminates the need for rate adapters and physical interfaces. With TBV, each physical interface is replaced with a transaction-level interface. Likewise, the rate adapter, required for ICE, is replaced with a protocol-specific transactor.

Unlike rate adapters, transactor models for the latest protocols are readily available off-the-shelf and are easily upgraded to accommodate protocol revisions. Synopsys today, for example, provides vast libraries of transactors for standard interface protocols as well as tools to enable the development of custom, proprietary transactors.

It is also possible to create an emulation-like environment by using transactors to connect the DUT to "virtual devices." A virtual device is simply a soft model of a device that runs on the host PC. A design using an HDMI transactor to test an HDMI interface, for example, would be able to leverage the same transactor to connect to an HDMI virtual display. This would allow the engineer to see the video output from the

DUT just as if it were connected to a physical monitor, without the need to physically connect to an actual monitor.

Some of today's virtual devices are bridges that provide connectivity to the actual physical device, but via the host PC (and a transactor). A USB bridge, for instance, connects a physical USB device, such as a memory stick, to the USB port of the host PC. As designed, the DUT only sees the USB drive and behaves accordingly once it detects the presence of the flash drive (for example, it installs a driver, if the OS is also being emulated).

An additional distinction of TBV is remote accessibility. As there are no physical interfaces connected to the emulator, a user can fully use and manage the emulator from anywhere in the world. When combined with virtual devices such as video displays, protocol analyzers, and audio processors, this distinction is very obvious.

A user from anywhere in the world can remotely view a virtual display. An ICE user, on the other hand, would need to be in the same room with the TV, which is cabled directly into the emulator. Remote accessibility is also available when connecting the emulator to an architectural model environment, such as Synopsys' Platform Architect and/or Virtualizer, which would also be done through a transaction-level interface.

A transactor consists of three parts: the front-end interface, a back-end RTL bus-functional model (BFM), and a physical communications channel.

The front-end interface is typically a behavioral model that runs on the host PC and interfaces to the testbench, usually through Direct Programming Interface (DPI) calls. It is written in C/C++, SystemC, or System Verilog. This piece converts high-level transaction-level commands and sends them across the physical interface protocol using either the Standard Co-Emulation Modeling Interface (SCEMI) standard or the Zebu Emulation Modeling Interface (ZEMI) language.

The interface channel between the front-end interface on the host PC and the BFM on the emulator may be implemented at the physical level using any high-performance interface standard such as PCI Express. A standard TBV protocol, such as SCEMI or ZEMI, runs on top of this physical interface. This protocol is transparent to the user.

The back-end hardware description level (HDL) BFM runs on the emulator and interfaces with the communica-

tions channel to send and receive transactions. Here, the transactions are converted to a wire-level interface that interacts with the DUT. Because the BFM resides on the emulator, it must be fully synthesizable. The BFM also must adhere to the target interface protocol and therefore must be fully functional.

So, as an example, the BFM would receive the high-level transaction command "read file" from the channel and convert it to the precise, cycle-accurate, wire-level handshake sequences that a physical direct memory access (DMA) controller, located within the emulated DUT, would need.

TBV IN THE DESIGN FLOW

TBV can be used throughout the flow, from the block level and beyond. Common applications include:

- Debug and verification of large blocks, subsystems, or complete SoCs
- Driver development or system-level verification when combined with virtualization
- Early hardware/software bring-up, including firmware, drivers, and OSs
- Full-chip power analysis and estimation
- System-level hardware/software co-verification or early software development when connected to prototyping hardware

CONCLUSION

TBV with emulation offers design teams a unique opportunity to accelerate SoC verification. By raising the level of abstraction of their verification environment, teams can achieve multiple orders of magnitude improvement in their verification performance. With TBV, designers have access to a platform that delivers megahertz performance for block- and system-level verification, as well as early hardware/software bring-up.

The use of transactors delivers all the benefits of in-circuit emulation without the challenges of rate-adapter availability and physical accessibility. When combined with virtual interfaces and system-level virtual models, TBV with emulation can be used throughout the verification cycle, with worldwide 24x7 access.

LAWRENCE VIVOLO is the product marketing director, emulation, at Synopsys. He holds a BS in engineering from California Polytechnic State University, San Luis Obispo, and an MBA from Santa Clara University, Santa Clara, Calif.