cādence®

De-Mystifying SCE-MI Transactors for Simulation Acceleration

Authored By Cadence Verification Acceleration Product Engineering and Product Marketing

To speed verification, electronic design automation (EDA) vendors provide off-the-shelf Standard Co-Emulation Modeling Interface (SCE-MI)–based transactors for publicly defined protocols and interfaces, such as USB, PCI Express, and Ethernet. To optimize these off-the-shelf transactors for proprietary protocols and interfaces, users need a basic understanding of transactors for simulation acceleration. Cadence emulator solutions provide a fast way to iterate through code revisions, providing results that are congruent with simulator results.

Contents

Introduction................................1

Scope...1

Accelerating the Verification Environment...............................2

Conclusion..................................8

Introduction

With multiple interfaces embodied in many of today's state-of-the-art SoCs and ASICs, the need to accelerate the verification of these complex systems and protocols with simple yet encompassing solutions is vital. The more these solutions can be based on standards, the better engineers can reuse their verification components throughout the verification lifecycle of their products, and for derivative products. In simulation acceleration, for publicly defined protocols and interfaces (such as USB, PCI-Express, Ethernet, etc.), electronic design automation vendors provide off-the-shelf standards-based transactors that help users quickly plug the transactors into their verification environment, thereby shortening the time to accelerate their simulation. However, for proprietary protocols and interfaces, users require a simplified, de-mystified understanding of transactors to create and optimize standards-based transactors for simulation acceleration. This paper describes a framework for a transaction-based component such that it can be accelerated by a hardware verification engine.

Scope

The bus interfaces of today are complex and difficult to verify, requiring many test sequences to ensure they are implemented correctly. Add to this the task of system-level verification using even larger datasets, perhaps including stimulus applied through these interfaces, and the challenge is clear.

With hardware-assisted verification solutions such as emulators, the system-level bugs (either hardware or software) can be found much earlier in the development and verification process. Simulation acceleration is a mode wherein the engineer's design under test (DUT) is mapped into the emulator to run faster, while the testbench (and any behavioral design code) remains running in the simulator or as a standalone process on a workstation. A high-bandwidth, low-latency channel connects the workstation to the emulator in order to exchange data between the testbench and the DUT. Overall performance is typically limited by the communications channel between the

emulator and the workstation and by the testbench execution time of the components running on the workstation. Transaction-based acceleration is a performance-optimized form of simulation acceleration in which transactions, or high-level messages, are sent to the emulator rather than via bit-by-bit, cycle-by-cycle exchanges. This reduces the traffic between the workstation and the emulator, allowing higher performances to be achieved and thus reducing the time to find system-level bugs that may take many cycles to become apparent.

Different hardware-assisted verification vendors have historically provided proprietary transactional interfaces in an attempt to optimize performance and functionality. The Accellera committee (www.accellera.org) has for some time been working to standardize transaction-based interfaces (or bridges) between simulators and emulators to enable interoperability. Since 2007, the committee has proposed version 2.0 of the Standard Co-Emulation Modeling Interface (SCE-MI), which encompasses three communications protocols: function based, pipes based, and macro based.

Rather than describe the SCE-MI APIs, this paper will outline: the layers of SCE-MI–based transactors; a component that decomposes or reconstitutes high-level messages to and from the bit-by-bit, cycle-by-cycle interface; and finally, performance optimization considerations for transactor development.

Transactors are the bridges that enable verification environments to decouple the testbenches from the nuances of the interface protocol. Many of the interface protocols are industry-standard, so time invested in developing a consistent methodology for transactor development enables transactors to be verified only once and reused in many future projects. This paper focuses on good general practices for partitioning a transactor, leveraging transactors in a complex verification environment, understanding the benefits of encapsulating complex protocols, and optimizing for performance and portability. Understanding transactors is the bedrock for creating accelerated verification components for any dynamic verification methodology, such as advocated by the Open Verification Methodology (OVM).

Accelerating the Verification Environment

The following sections describe the methodology that takes a logical verification environment to an environment that is acceleratable.

The logical environment

Figure 1 represents the logical view of a verification environment. It is composed of two logical blocks: the design under test (DUT) and the testbench. The DUT is ultimately going to be manufactured. The testbench manages the verification environment. All communication between the testbench and the DUT is accomplished via the DUT's signal-level interfaces.

The DUT encapsulates all of the IP and RTL blocks required to implement the desired functionality. For the purposes of verification using simulation acceleration, all of the blocks internal to the DUT must be emulatable. Exceptions are made for support IP that exists as a stand-in for any non-emulatable blocks (such as pure analog blocks). Any synthesizable IP remains in the DUT.

The testbench encapsulates all of the verification components required to apply stimuli, check DUT responses, and manage the environment. A minimal testbench comprises a stimuli generator, master transactor, monitor transactor, and response checker.

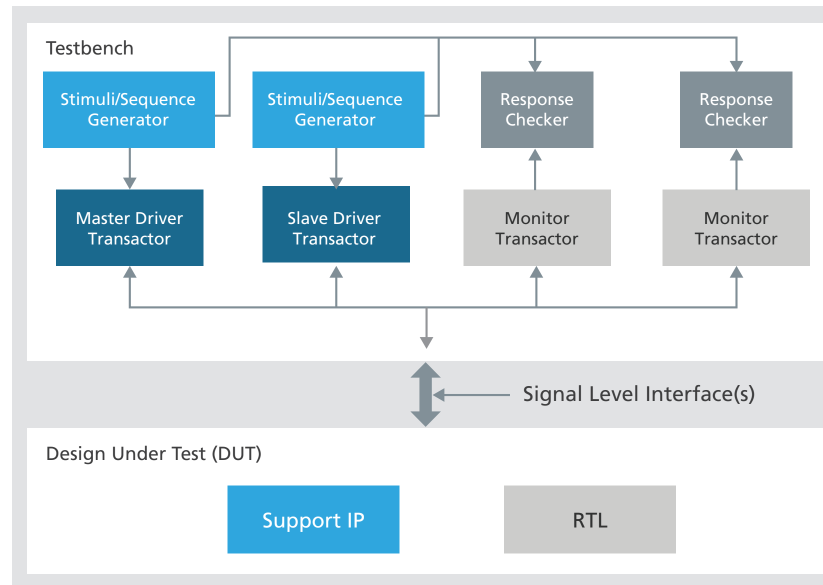

Figure 1: Logical view of a verification environment

The transactors translate transactions to the signal-level interfaces of the DUT. In general, there will be at least one master driver and one monitor transactor attached to each unique DUT interface.

The transactor types have the following primary characteristics:

- Master driver: Initiates activity on a signal-level interface by translating sequences of transactions to signal-level activity
- Slave driver: Responds to activity on a signal-level interface (like a memory) by translating signal-level activity and responding with sequences of transactions as a result
- Monitor: Observes activity on a signal-level interface for the purpose of reporting by translating signal-level activity to transactions

It follows that a master/slave/monitor transactor set specializes in a particular interface. For a transactor set, no specific characteristics are embodied with respect to stimuli sequences other than a protocol-correct implementation of the signal-level interface. In this manner, the transactors can be used with any test sequence with any DUT that utilizes the transactor's signal-level interface.

The remaining verification components are the stimuli/sequence generator and response checker, which can interface with one or more transactors. A stimuli generator will generate a sequence of stimuli transactions, representing a specific test. A response checker will verify the sequence of result transactions, based upon a set of stimuli. In general, a stimuli generator will be connected to a response checker so that it knows the cause of the effect observed by an attached monitor transactor. Although this implies that the driver and monitor transactors are connected to the same signal-level interface, this is not necessarily the case. Changing test sequences is as easy as substituting the stimuli generator/response checker pairings for the affected interfaces.

The verification environment described above is suitable when conventional software simulation tools are used. However, it is not optimized to provide maximum simulation performance through the use of emulation technology. It is necessary to separate the acceleratable aspects of the verification environment from the non-acceleratable. The key is to construct transactors so that they can be used in an accelerated, high-performance verification environment.

The acceleratable transactor

An acceleratable transactor (see Figure 2) consists of three main layers:

- Proxy model with transactional interface
- One or more communications channels
- Bus functional model with signal-level interface (written in HDL)

Together, the three layers form the physical representation for transactor types: master, slave, and monitor.

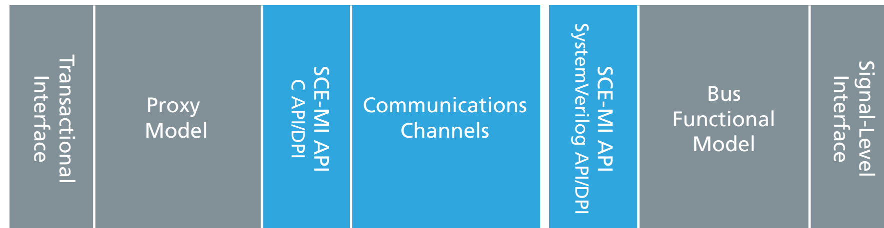

Figure 2: An acceleratable transactor

The proxy model takes a transaction provided via the transactional interface, converts it to messages, and then passes it to a communications channel interface. In general, no state-specific information is transmitted. Any transaction processing, other than message packing and unpacking, should only be performed by the proxy model if it improves performance.

The communications channel(s) provide the means to buffer and transfer messages to and from the proxy model and bus functional model (BFM). These communications channels are formed using SCE-MI.

The BFM receives messages from the communications channel and converts them to appropriate state changes on the signal-level interface. It is the responsibility of the BFM to implement the clock-by-clock specific protocols of the signal-level interface (AXI, USB, PCI Express, etc.).

This process can of course operate in reverse, where the BFM observes the signal-level interface and creates messages to be sent to the proxy model, which converts the messages into transactions to be sent via the transactional interface. A monitor transactor operates in this manner.

The proxy model and BFM pair must be designed for maximum performance through the best utilization of workstation, emulator, and communications channel resources. This will be examined further in a later section.

It is the SCE-MI capabilities that enable a transactor to be divided into two physical parts: the BFM resides in the emulator and the remainder (the proxy model) resides in a workstation. It is this partitioning that allows a transactor to be accelerated.

The physical environment

By employing the acceleratable transactor, it is now possible to construct a verification environment that takes advantage of the performance provided by emulator technology. The structure of the logical verification environment is altered for acceleration by partitioning it into that which is acceleratable and that which is not.

Figure 3 illustrates the physical partitioning of a logical verification environment when using an emulator. All testbench aspects that cannot be accelerated (stimuli generators, response checkers, proxy models, support IP) will be processed by the workstation. Acceleratable testbench aspects (BFMs) and the DUT are placed in the emulator. From a verification environment point of view, the logical representation of the transactor has not been affected by its physical partitioning. The horizontal dot-dash box encloses the three layers of an acceleratable transactor as it exists within the accelerated verification environment.

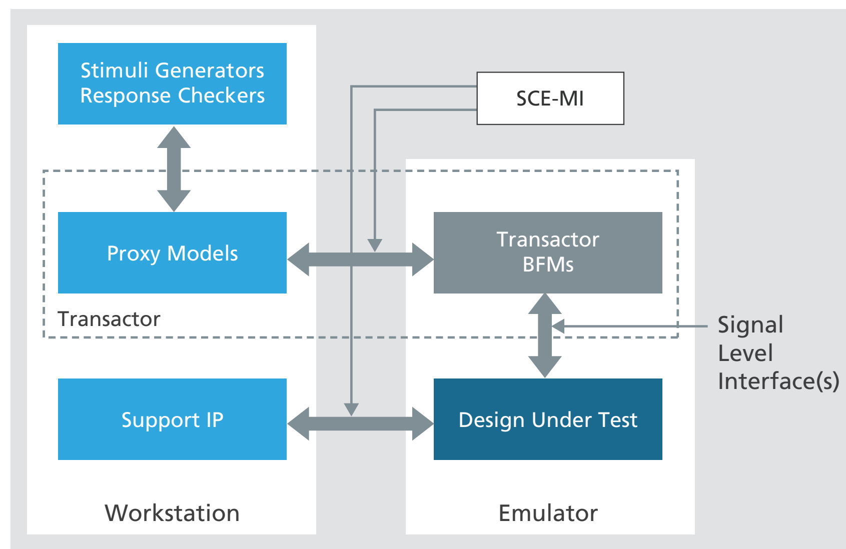

Figure 3: Physical partitioning of a logical verification environment using an emulator

This environment is applicable for any tools or applications (such as an event-based simulator) required by the stimuli generators, response checkers, proxy models, and support IP.

Testbench considerations in accelerated environments

There are several things to take into account when using an acceleratable verification environment: general, determinism, time, and concurrency.

General

Placing the verification environment into a known state could require several operations or steps. These steps can include resetting the DUT; initialization of DUT memories and/or registers; initialization of support IP; and transactor initialization and configuration. The order that these steps are performed depends upon the requirements of the test. However, it is usually necessary to initialize and configure the transactors first, since it is via the transactors that the other steps can be accomplished.

Determinism

Strict determinism requires that the stimuli and responses occur on the same clock edge (or emulator time) for each run, involving identical stimuli sequences. A deterministic DUT and testbench provides the simplest environment for verification and debug because responses are predictable. If the DUT or testbench is not deterministic, it will be the responsibility of the verification environment to make allowances by buffering the expected responses and creating checkers that can handle responses that are out of order or at unpredictable time steps.

Time

Emulator time is the domain in which the DUT operates, similar to a software-based simulator, and can be suspended when required. This allows SCE-MI communications to consume zero emulator time and be deterministic.

Wall clock time is the domain in which humans and workstations operate and cannot be suspended. The amount of wall clock time required to complete a function is dependent on OS/communications latency and workstation speed/loading. SCE-MI communications consume wall clock time subject to these conditions.

Concurrency

Concurrency is created when one or more verification components operate truly independently within the environment. An example where this is sometimes useful is a stimuli/sequence generator controlling the generation and application of stimuli via a master transactor independently of the DUT. Such parallelism can be a performance boost.

Concurrency created within the emulator's time domain (through the use of Verilog procedural or VHDL process blocks) will remain deterministic, as long as the concurrent process' algorithm is deterministic—even race conditions behave consistently within the emulator. But a component in a process or thread not controlled by the emulator is operated outside of the emulator time domain and can prevent the verification environment from being deterministic.

Performance

It is possible to have a sophisticated high-level language testbench and still achieve in-circuit or synthesized testbench performance levels. It does, however, require careful attention to performance factors.

The first and foremost performance factor is the testbench overhead. Remember that the DUT evaluation is running much faster in emulation than in pure simulation. A testbench that is a minor portion of the overall time in a simulation environment can easily be the performance limiter in an accelerated environment. Figure 4 shows the dramatic effects of testbench overhead. Consider a lightweight testbench that only takes 1% of the wall clock time in a software simulation. By moving the design into an emulator, the design simulation time becomes negligible and the overall speedup from the pure simulation run is about 100x. However, if the testbench consumes 50% of the wall clock time in a software simulation, then the overall speedup by moving the design to an emulator is, at best, only about 2x.

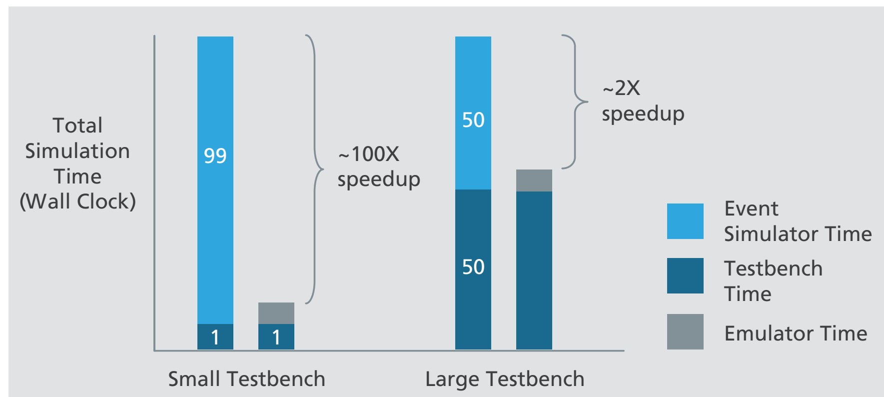

Figure 4: Effects of testbench overhead

Typically a testbench is unable to perform complex computations for each transaction or keep pace with the accelerated portion of the design. Consider the amount of time it takes for a transaction to be applied to and evaluated by a DUT in the emulator. In the same amount of time, a testbench running on a workstation can only execute 10s to 100s of instructions. As an alternative to dynamically running complex algorithms during a test, consider doing pre-computation of stimulus or post-processing of results. Also think about having the BFM do some of the transaction processing beyond the basic tasks of mapping the transaction data to and from the DUT signal-level interface.

Consider using concurrency, which gives you the ability to overlap testbench computations with evaluation of the DUT in hardware. Concurrency is most appropriate for very loosely coupled testbenches and DUTs, such as streaming interfaces. In this case the computation of the two sides as well as some of the communication overhead can be overlapped.

However, for more frequently interacting testbenches and DUTs, concurrency can actually be a hindrance. Since both sides are running concurrently, one or both sides may end up waiting to synchronize and exchange data. Also be aware that there is an increase in the difficulty of implementation and debug since you are no longer guaranteed congruent results with simulation, nor are you assured of congruent results between multiple simulations.

The next performance factor to consider is the interaction rate between the testbench and the DUT. There is a significant latency to synchronizing between the workstation and the acceleration hardware. Think about ways to maximize the amount of simulation time that can occur before synchronization must happen. Transaction methodology in itself is a key element. A transaction, by definition, is compressing a sequence of actions on a physical interface occurring over time into a single entity. Using the buffering capability in the SCE-MI pipes-based API to send multiple transactions can be useful to minimize synchronizations as well.

Figure 5 illustrates the effect of synchronizations on performance. It is wise to categorize the effects of synchronization before embarking on architecting your transactors:

- Left column – Performance is severely affected by synchronizations between testbench and the emulator; improve performance by increasing the use of transactors or buffering
- Middle column – Performance is pretty good; improve performance by decreasing the synchronization rate, but consider carefully all other performance factors
- Right column – Performance is not affected by testbench synchronizations; additional optimizations will only add complexity without significantly increasing performance

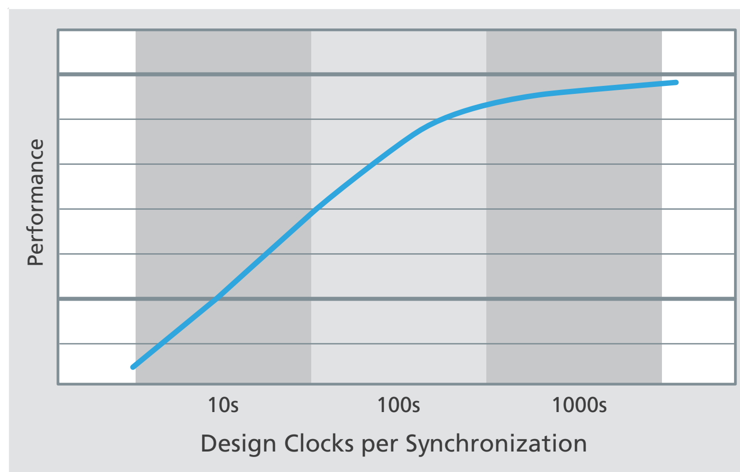

Figure 5: Effects of synchronization on performance

Other performance factors to consider are the absolute amount of data being passed and the shape of that data. Although the communications channels in each hardware configuration can vary, it is usually best to send as much data as possible through the channel at one time. Aggregate as much data as you can into a single transaction and use channel buffering mechanisms when the nature of your transactor allows for batch exchanges of transactions. Use the BFM to encode or decode data in a transaction. The emulation hardware is very efficient at bit manipulations, so move some of the low-level processing to the BFM.

Conclusion

For protocols and interfaces that are publicly defined (USB, PCI Express, Ethernet, etc.), EDA vendors provide off-the-shelf SCE-MI–based transactors that help users quickly plug the transactors into their verification environment, thereby shortening the time to accelerate their simulation. However, for proprietary protocols and interfaces, users require a simplified, de-mystified understanding of transactors to create and optimize SCE-MI–based transactors for simulation acceleration.

To build and test performance-optimized SCE-MI transactors, the emulator of choice must offer salient features to complement the development. With fast compile times akin to simulation compile times and with a debug environment that offers full design visibility at both signal and transaction levels, the Cadence® emulator series and software offer a significant edge over competing solutions. Since transactor developers will typically need to iterate through several revisions of their code development in the course of a day, testing them on an emulator that enables quick turnaround is beneficial. Moreover, Cadence emulator results are congruent with simulator results. As such, when bugs are exposed in an accelerated run, they can be replicated and debugged by users with access to simulators only. For instance, IP developers (especially third party) may have not access to the emulator, but can reproduce bugs detected in a system-level verification environment.

In addition, Cadence emulators are well-versed with practical applications of SCE-MI in multiple customer projects and in their own verification IP portfolio development. Cadence verification environments integrate tightly with Cadence Incisive® Enterprise Simulator for testbenches built with Hardware Verification Language (HVL) content such as SystemVerilog or e, with standalone applications written in C/C++, or with third-party simulators using unique Cadence Direct C capabilities.

Cadence is transforming the global electronics industry through a vision called EDA360.
With an application-driven approach to design, our software, hardware, IP, and services help customers realize silicon, SoCs, and complete systems efficiently and profitably. www.cadence.com

©2011 Cadence Design Systems, Inc. All rights reserved. Cadence and the Cadence logo are registered trademarks of Cadence Design Systems, Inc., All rights reserved. Cadence and the Cadence logo are registered trademarks of Cadence Design Systems, Inc., All rights reserved. Cadence and the Cadence logo are registered trademarks of Cadence Design Systems, Inc. 21392 12/11 MK/DM/PDF