INTRODUCTION

This paper describes a unified flow for both hardware-assisted RTL verification and pre-silicon validation of hardware/software integration and how we set up the associated testbench environment by leveraging verification industry standards for reuse (UVM) and co-modeling (SCe-MI 2). Deployment of this flow was accomplished in a customer setting by first combining a mainstream, transaction-level verification methodology — the Universal Verification Methodology (UVM) — with a hardware-assisted simulation acceleration (also known as co-emulation) platform. The necessary testbench modifications incorporated to make this combination are generally nonintrusive and required no third-party class libraries; pertinent verification components from the customer environment were hence readily reusable in the pure simulation environments, across different designs using the same block, and for different verification groups.

Next to offering substantial speed-up for verification in terms of simulation cycles per second, the common-source SystemVerilog and UVM acceleration platforms outlined above has subsequently been leveraged also for software validation. The hardware-assisted simulation acceleration factor of two to three (or more) orders of magnitude have made it practical for the software engineers to begin co-validating the software with the hardware far in advance of silicon. Clearly, the benefit of pre-silicon validation in terms of early detection of hardware/software integration issues boosts the efficiency of post-silicon validation as well.

The unified verification and validation flow has enabled a seamless transition between RTL design verification and software/firmware validation. It significantly reduces the turnaround time for time-consuming yet essential tasks, including debugging and regressions. It takes advantage of very fast emulator performance to handle longer and more tests to cover more defects and to discover more design bugs. In essence this unified flow has eliminated the productivity and quality penalties associated with creating and maintaining different verification and validation platforms.

TESTBENCH ARCHITECTURE

RTL design verification

A conventional verification environment has both synthesizable and non-synthesizable components instantiated in a single top module, as shown in Figure-1. This hinders running the testbench in a co-simulation setup, where two different physical devices are involved: a hardware emulator and a simulator. A recommended approach is to create an acceleratable testbench that is partitioned into two tops: HDL_TOP and HVL_TOP. HDL_TOP has all the synthesizable components instantiated in it. HVL_TOP contains all unreplatable behavior as shown in Figure-2. Synthesized HDL_TOP runs on the hardware accelerator and HVL_TOP runs on the simulator.

The HDL and the HVL top-level models communicate at the transaction level. This communication is enabled by using a SystemVerilog virtual interface based method and/or a core-proxy based method.

IMPLEMENTATION FLOW

We bridged these partitioned domains using a SCe-MI compliant, high-performance, transaction-level communication link between the hardware and software provided by the Mentor Graphics Veloce/TBX solution. This allowed us to accelerate the time portions of the testbench and the DUT in the Veloce emulator without affecting the untimed UVM domain.

A virtual interface is a SystemVerilog variable that can hold a reference to a concrete SystemVerilog interface instance. A variable of a virtual interface type can be given a value (i.e., it can be made reference to an existing interface instance) by assigning to it the hierarchical path name of the given interface instance [6]. The Mentor Graphics Veloce® emulation platform supports a synthesizable transaction interface that provides communication between emulator and testbench. Transactor interfaces encapsulate synthesizable SystemVerilog tasks and functions. A driver calls a function or task using a reference to a synthesized SystemVerilog interface to access DUT pins and signals. Similarly, a monitor waits on a function call from a transaction interface. All accesses to DUT pins and signals are thus strictly confined to the HDE; partition. All such functions and tasks are restricted to synthesizable data types.

A co-proxy is a C++ core based model containing communication semantics between HDL and C, as defined by SCe-MI 2 (2). A SystemVerilog API wrapper class connects a C based proxy to the rest of the testbench. The proxy class maintains handles to the compiler in the synthesizable transactors and uses DPI function calls or SCEMI pipes to communicate with these transactors [10]. In this approach, C-based proxy class functions provide APIs that can be used by a SystemVerilog or SystemC core to communicate with the DUT.

A conventional software validation platform usually involves an FPGA prototyping platform with a JTAG connection to a software debug probe. Typically, a dedicated team is responsible for partitioning the system on chip (SoC) RTL for a specific hardware platform to meet capacity and protocol constraints. As more and more processor cores become part of these processes, there is a growing requisite for hardware/software co-validation. Earlier access to complete RTL, maintaining system bus connectivity across sub-modules and protocol peripherals, is an essential step of the verification flow.

A preferred approach for hardware/software co-validation is to leverage the verification testbench to access the same RTL as used by design verification engineers. Based on the testbench modeling method described in the next section, a user can access the SoC system bus both with a UVM testbench or with a C API embedded software model to access the complete RTL. Figure-3 demonstrates how a virtual machine communicating with the Veloce emulator to validate software driver development using a proxy-based transactor. Such integration provides the complete SoC RTL in the hardware accelerator with full visibility.

OTHER CONSIDERATIONS

To effectively implement the unified testbench for simulation and acceleration, we adhered to the following coding guidelines:

• Delays are not allowed in the testbench code. To achieve best performance, all code on the HVL testbench side should be untimed, and all timed constructs removed.

• There should not be any direct signal access from the HVL side. All communication must be transaction-based. To access individual signals indirectly, DPI transactors were used.

• Memory models can remain as behavioral Verilog code. The Veloce compiler can infer structural memories from behavioral Verilog models.

CONCLUSIONS

The adopted co-emulation flow and resulting unified verification and validation platform has made it possible for SoC design verification and software validation teams to use the same RTL image for their respective test scenarios. Moreover, the software validation team can now have much earlier access to the RTL code than in previous projects of similar scope. For the case study at hand, a platform for validation was available to the software team about four months sooner.

Further work is focused on integrating assertions data from the emulator in a common database to support coverage-driven verification using an emulator. With the standardization of the Unified Coverage Interoperability Standard Database (UCISDDB) [5], engineers can collect coverage data in one common database from simulation and acceleration platforms. Thus, providing the ability to use software generated vectors to assert RTL coverage metrics.

With the speed of an emulator and a combination of directed tests as well as real-world scenarios, verification engineers will be able to validate the SoC more holistically. As simulation acceleration techniques get adopted more broadly in the industry, advanced standards-based verification methodologies such as UVM should increasingly accommodate requirements from hardware-assisted verification testbenches.

ACKNOWLEDGMENTS

E. van der Schoot, A. Sahx, A. Grug, S. Krishnamurthy, “Off to the races with your accelerated SystemVerilog testbench.” DVCon 2011


Unifying Hardware-Assisted Verification and Validation Using UVM and Emulation

Hemant Sharma, Hans van der Schoot, Achutam Murarka

Mentor Graphics Corporation

Figure-2: Two-top testbench for HW-assisted acceleration

Figure-3: Two-top testbench for SW validation

Figure-1: Traditional simulation testbench

Reusing the testbench for software validation

A typical SoC has a single processor managing multiple subsystems or multiple processors designed to perform specific tasks — all communicating via a system bus. A proxy-based transactor was developed to provide a master and slave interface to the system bus. Such an implementation provides a UVM testbench interface that can be used with a SystemC® or UVM testbench. Software teams usually have a system model available for early code development. SystemC® has emerged as the language of choice for these models. Hence, we focused on language interoperability of the transactor; i.e., for portability between SystemC models on the one hand and SystemVerilog testbenches on the other hand. Figure-3 shows how the testbench is used to verify system environment setup for software validation. There are various hypervisor and virtual machines available to emulate the processor architecture. One such virtualization software package is QEMU, which can run in conjunction with the Mentor Veloce emulator. Software users can run their executable code in virtual machines that communicate with the RTL via a proxy based system bus transactor. As a result, the same RTL image is available for verification and validation with full visibility to debug.