FlexBench: Reuse of Verification IP to increase Productivity

Bernd Stöhr, Michael Simmons, Joachim Geishauser
Motorola, Munich, Germany
{Bernd.Stoehr,Michael.Simmons,Joachim.Geishauser}@Motorola.com

Abstract

This paper presents FlexBench, which is a complete framework for SoC verification at the Module and SoC level, both with and without embedded processors. The focus is to increase the productivity of the verification engineer by providing a framework to reuse verification IP, which includes parts of the testbench and the test stimulus.

1 Introduction

It is pretty well accepted that the biggest challenge of modern complex chip design is verification-i.e. finding all bugs and problems prior to tape out [1][4]. By raising the abstraction level from pin-level to higher levels, the productivity of verification engineers can be significantly increased. Instead of driving the pins of a design under verification (DUV) directly from a testbench, an additional layer is created between the two. This layer is implemented as a set of Drivers (D) and Monitors (M), which do the actual driving and monitoring of the DUV pins.

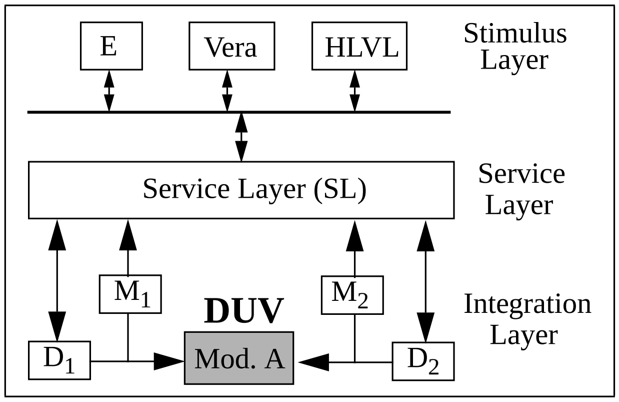

Figure 1: FlexBench Architecture

2 FlexBench Architecture

The FlexBench architecture is a unique architecture built on the concepts that we have discussed so far. The architecture can be divided into roughly three layers (see figure 1):

- Stimulus Layer: This layer is the “user interface” to the DUV. It provides a Transaction level interface to the Drivers and Monitors. Due to a defined interface any stimulus language which provides a Verilog interface can be used.
- Integration Layer: This layer includes all of the Drivers/Monitors in the simulation.
- Service Layer: This layer provides the connection between the Stimulus Layer and the Integration Layer, as well as some “system services.

The stimulus language in FlexBench is C with additional libraries, and is called the High Level Verification Language (HLVL). These libraries provide following functionality:

- Support for arbitrary length 5-state numbers (0,1,x,Z,”Don’t Care”)
- Transaction interface with coverage
- Threads, Semaphores, Regions and Mailboxes
- Constrained random number generation
- Debug support and message handler

The reuse of Module-level stimulus on SoC level is solved with Stimulus partitioning:

- StartUp: Module level specific initialization code goes here.
- Stimulus: The main body of the test.
- ShutDown: Module level specific shutdown code goes here.

3 References

[1] Kuhn et. al.; *A framework for object oriented hardware specification, verification, and synthesis* DAC 2001 pp. 413-418

[2] http://www.verisity.com/

[3] http://www.open-vera.com/

[4] Albin, K; *Nuts and bolts of core and SoC verification* DAC 2001, pp. 249-252