

Yassine Eben Aimine explains the considerations for a functional verification approach to debugging complex SoCs

Efficient constraint solving and debugging of SoC designs

For years, hardware verification languages (HVL) were a specialist topic. The burden of moving away from the main Hardware Description Language (HDL) to a software-oriented framework proved to be an obstacle to wide deployment.

Offering the capabilities of powerful HVLs within the same HDL simulator led to the mass adoption of SystemVerilog.

SystemVerilog allows for bit-level and word-level quantities to be assigned a random value under a set of constraints. With this, a single verification testbench can be constructed to generate verification scenarios that validate most of the functional specifications of the device under test.

In theory, the same testbench creates a different testcase for a different seed. In practice, this requires an efficient solver to generate random solutions with good distributions.

Moreover, complex SoCs integrate rich functionalities requiring very large stimulus space. This, in turn, requires complex constraints involving hundreds of variables. Debugging such constraints requires innovative technologies capable of visualising the solver operations over declarative constraints.

A functional verification approach can address these issues with a multi-engine constraint solver and automatic constraint profiling and tracing.

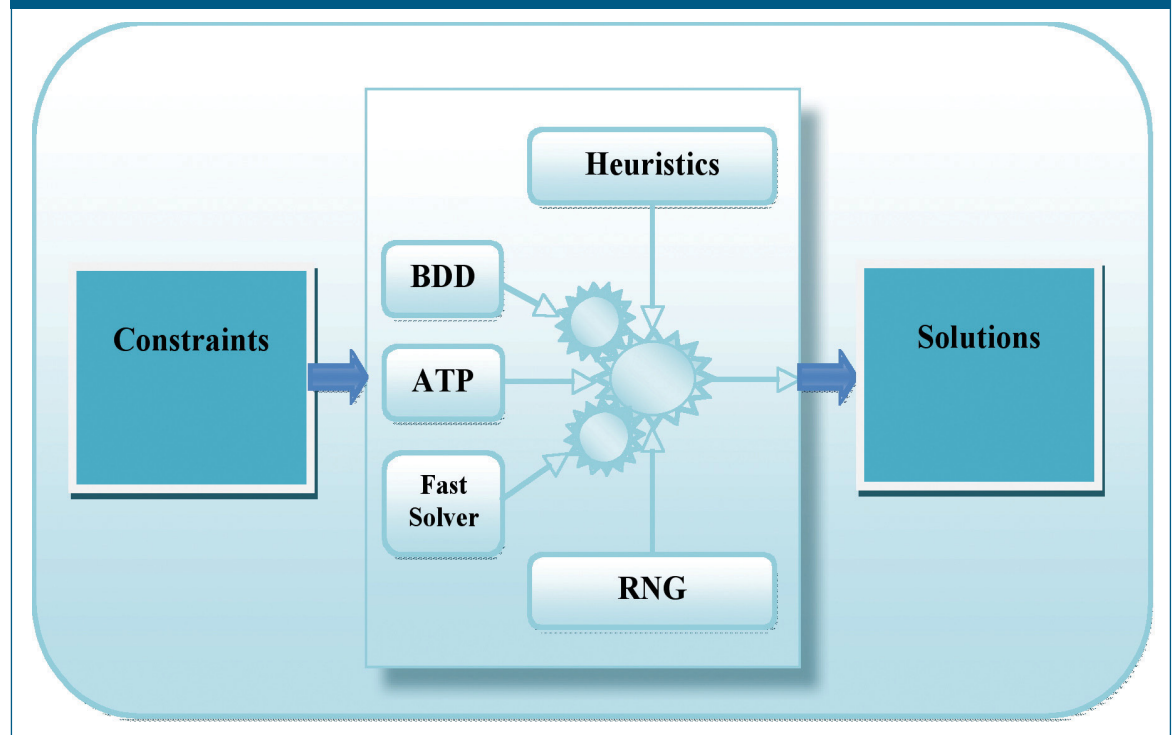
SystemVerilog allows for declarative constraint programming. Constraints can be simple or complex. They can specify range constraints, logical and arithmetic relationships. They can be defined over simple or compound variables.

At the core of this approach will be a multi-engine solver, comprised of a random number generator, a binary decision diagram (BDD) engine, a satisfiability engine and a test pattern generator.

BDD solver

A binary decision diagram (BDD)

FIGURE 1: MULTI-ENGINE SOLVER FOR VERIFICATION



is a graph-based data structure that represents a Boolean function, which models the constraint satisfaction problem.

Constructing the BDD allows for all constraint solutions to be computed at once.

To compute a solution, VCS picks a path starting from the root and ending at the node 1. Randomness is introduced through the branching decisions along the traversal.

ATPG solver

Automatic test pattern generation (ATPG) is well known as the process of generating test vectors to detect manufacturing faults. The basic algorithm uses a search-based method to find an assignment to the circuit inputs that lead to a net being applied of a logic one or zero.

In this solver mode, given a constraint satisfaction problem, it is possible to construct a single output

logic circuit. After reducing the size of the circuit, the branch-and-bound algorithm is applied to find input values that drive the output to a logic one.

In contrast to a BDD approach, the ATPG mode is a search algorithm that finds a single solution per invocation.

Fast solver

SystemVerilog raises the level of abstraction at which testbenches are written. In these high-level descriptions, the use of multibit variables, such as integers, becomes common. In many situations, it is best to reason at the word level to solve the constraints.

One approach is to use a technique known as integer linear programming (ILP) to generate constraint solutions to multibit variables, without resorting to expanding them to bits. This results in faster runtimes.

Constraints debug

With the ever increasing complexity of designs comes an ever more complex set of testbench constraints involving dozens, if not hundreds, of variables.

Debugging declarative constraints failures and fine-tuning performance can be tedious without a dedicated, solver-aware debugger.

Writing complex testbenches can be made easier by implementing a simulation infrastructure, which contains a constraint solver that is flexible and powerful. The solver uses multiple engines to solve highly complex constraints. Debug of such constraints is made simpler with automatic profiling and tracing. ●

Yassine Eben Aimine works for Synopsys

Synopsys
www.synopsys.com

