VEasy: a Functional Verification Tool Suite

Samuel Nascimento Pagliarini and Fernanda Lima Kastensmidt
{snpagliarini, fglima}@inf.ufrgs.br

Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Programas de Pós-Graduação em Microeletrônica e Computação
Porto Alegre, Brasil - 91501-970

Abstract

This paper describes a tool developed specifically for aiding the process of Functional Verification. The tool has three main built-in modules: a Verilog Linter, a Verilog Simulator and a Graphical User Interface for Testbench Automation. On top of these modules there is a methodology for collecting and analyzing functional and structural coverage. Results show that the built-in simulator enables a higher number of cycles per second while the user interface allows the creation of complex test scenarios.

1. Introduction

The primary goal of Functional Verification (FV) is to establish confidence that the design intent was captured correctly by the implementation [1]. However, the continuous increase in terms of the number of transistors per chip is resulting in a diminished validation effectiveness. Simulation is getting more expensive and providing less coverage [2]. FV strives to cope with that complexity increase trend but some of the related challenges are overwhelming. So far those challenges have been addressed with methodologies and Electronic Design Automation (EDA) tools but there is a claim for more innovation and automation improvement.

This paper describes and compares VEasy, an EDA tool suite developed by the author. VEasy aim is to be a FV solution, including a simulator and a testbench automation interface. This paper is organized as follows: Section 2 explains the tool in detail, including the possible work-flows while Subsections 2.1 and 2.2 deal with the linting and simulation built-in modules. Subsection 2.3 explains the methodology used for creating complex test scenarios. The different types of functional and structural coverage are shown in Subsection 2.4. Finally, Section 3 provides some concluding remarks.

2. VEasy and the work-flows

The tool has two distinct work-flows: the assisted flow and the simulation flow. Fig. 1 illustrates the assisted mode. This flow starts when the Verilog [3] description of the Design Under Test (DUT) is parsed and analyzed. If the analysis is not successful the user must fix the errors reported by the linter before continuing. From that same input, the interfaces (i.e. Input and output signals) and special signals (i.e. clock and reset) are automatically extracted. This information is sufficient to build a template of a verification plan. The template is the input of the simulation flow.

Fig. 2 illustrates the tool simulation flow. Initially, a verification plan file is loaded. This file contains all the information that is required to generate and simulate the test scenarios that the user creates through the Graphical User Interface (GUI). VEasy then is ready to create a simulation snapshot, combining the circuit description and the test generation capabilities. The use of a golden model is optional. All the generated code is ANSI-C [4], which allows it to be used in the majority of platforms and compilers. After the simulation is complete the tool automatically collects the coverage results, saves them into the verification plan and provides this info for the user analysis. If suitable a new simulation round may be started in an attempt to reach coverage holes. The verification plan file used by VEasy is actually a complete view of the verification effort. It includes the traditional lists of features and associated test cases but it also contains simulation and coverage data. This approach makes it a unified database of the current verification progress.

2.1. VEasy as a Verilog RTL Linter

Linting [5] guarantees that the input is written using strictly Register Transfer Level (RTL) Verilog constructions. A series of items are checked for RTL compliance. Passing all those checks means that the code may be converted to a C-like version.
2.2. VEasy as a Verilog RTL simulator

FV relies on fast and accurate simulation. In order to improve the number of cycles simulated per second, VEasy integrates the test case generation (i.e. the generation of inputs that build a certain test case) with the circuit description. The circuit description portion of the simulation snapshot is obtained from extracting three profiles from the Verilog code: the combinational logic, the reset sequential logic and the regular sequential logic. The combinational logic simulation is performed using a signature-based method. A pseudo code of the method is given in fig. 3.

Algorithm 1: Combinational logic simulation

<table>
<thead>
<tr>
<th>Algorithm 1: Combinational logic simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data: S such that S contains all combinational signals.</td>
</tr>
<tr>
<td>Result: Frozen S after a simulation cycle.</td>
</tr>
<tr>
<td>begin</td>
</tr>
<tr>
<td>2 LocalSignature ← ∅</td>
</tr>
<tr>
<td>3 Signature ← ∅</td>
</tr>
<tr>
<td>4 do_it_again :</td>
</tr>
<tr>
<td>5 execute update logic on S</td>
</tr>
<tr>
<td>6 foreach s ∈ S do</td>
</tr>
<tr>
<td>7 LocalSignature[s] ← s</td>
</tr>
<tr>
<td>8 end</td>
</tr>
<tr>
<td>9 if LocalSignature = Signature then</td>
</tr>
<tr>
<td>10 return</td>
</tr>
<tr>
<td>11 end</td>
</tr>
<tr>
<td>12 else</td>
</tr>
<tr>
<td>13 Signature ← LocalSignature</td>
</tr>
<tr>
<td>14 goto do_it_again</td>
</tr>
<tr>
<td>15 end</td>
</tr>
<tr>
<td>16 end</td>
</tr>
</tbody>
</table>

The signatures used are arrays, sized according to the number of signals being updated in the combinational logic. A signal may be a primary output or a internal wire or register. If the signature is the same for two consecutive evaluations then the combinational logic is considered to be frozen since not a single signal changed from one evaluation to another, which can be observed on line 9. The update logic (line 5) is very similar to the original input since C and Verilog have similar syntaxes and operators. The main difference resides in the fact that C has no direct single bit access. This issue is resolved by using masks and logical operators (and/or) to set or clear specific bits. This method reflects the combinational logic of a circuit where the signals have a switching behavior.

The reset sequential logic simulation is trivial: all resettable signals will receive the determined reset values. On the other hand, the regular sequential logic simulation must provide the concurrency of assignments as if the clock edge were reaching all the signals at the same time. For that matter, a method that uses local copies of the signals was developed. Such method works by performing all the sequential update logic assignments at the local signal copies but using the actual signal values, i.e. no actual signal is written until the logic has evaluated. After the logic is evaluated the local signals are written on the actual signals. Therefore the order in which the signals are assigned is no longer important.

Each of the profiles is built into a C function. The simulation process will repeatedly call these functions until the desired number of simulation cycles is reached. Separating the reset behavior from the regular sequential logic behavior saves some simulation time. A set of simple circuits was defined for the purpose of comparing the speed of the simulator. Fig. 4 shows those results, where diffrst is a d-type flip-flop with reset (negative edge), adder in an 8-bit adder with registered outputs, fsm is a simple FSM that has 8 states and performs different 16-bit data operations on each state and t6507lp [6] is a 8-bit micro-controller with 100 opcodes and 10 addressing modes.
All simulations of fig. 4 were done using 10 millions of clock cycles. The reset signal was asserted only during the first simulation cycle. The other signals were generated every cycle with a random value. Commercial I is a simulator from a major vendor in the ASIC domain. Commercial II is a simulator used mostly in the FPGA domain. Icarus Verilog [7] is a free software. The scale on the Y axis of Fig. 4 is logarithmic. When compared with the simulation times of Commercial I, VEasy performs, on average, the same simulation within less than 5% of the time Commercial I requires.

2.3. VEasy and the Testbench Automation

The main feature of the tool is the testbench automation. Using a strong GUI the user is able to drag-and-drop sequence to build larger sequences. One example of that operation is shown in Fig. 5, where the user is dropping a layer2 sequence item (press_and_releaseY) into a layer3 (correct_password) list of sequence items. The possibility of combining more and more layers allows the construction of sophisticated test cases that are of particular interest for the functional verification of a design. There are only a few rules that must be observed:

1) Sequence items of layer0 are the only ones capable of interfacing with the design.
2) Sequence items of layer0 will always generate values for all the inputs of the design, whether they are constrained or not.
3) If a member is not under any constraint then it is assigned a random value within its possible values.
4) Sequence items of layer0 are the only ones that can make the simulation advance in time.
5) The only communication channel between two layers is through logical members. All data exchange relies on this approach.
6) All members must be uniquely identified to allow any layer to use them unambiguously.
7) Each sequence must have at least one sequence item, except for layer0 sequences which do not have a list at all.

The process starts by defining at least one layer0 sequence that contains all of the physical inputs of the DUT. To build a layerN sequence it is only required that the user adds at least one layerN-1 sequence item. It is also possible to add logical members into sequences of all layers. Fig. 6 contains an example of a possible layering. The example shows a sequence called top of a layerN. This sequence has two logical members (memberA and memberB) and each member is constrained with a set of rules. This same sequence has four items in its list of sequence items. One of these items is referred as main, which is a sequence from layer0. This sequence item in particular has no list of items since it is in the bottom of the hierarchy. Yet, since this is a
sequence from layer0, it contains a physical member referred as phy_member. As mentioned, each member of a layer, either physical or logical, might be constrained using rules. Currently the tool supports 7 types of rules:

- Keep value less than (<) or greater than (>)
- Keep value less or equal than (<=), or greater or equal than (>=)
- Keep value equal to (==)
- Keep value ranged between ([a:b]) or in a list of possible values ([a,b,c])

Fig. 6 – Layering example.

2.4. Coverage Methodology

The quality of the verification relies on coverage metrics, either functional or structural [8]. VEasy has integrated three different metrics that are based on structural coverage: block coverage, expression coverage and toggle coverage.

On the functional coverage side, VEasy allows coverage of inputs and outputs. The output coverage is performed directly on the primary outputs of the design. The input coverage, on the other hand, may be performed on the primary inputs or using specific logical members of the layers. The user must choose such members manually. In another words, this allows the user to define the functional coverage metrics of interest.

3. Conclusion

Design verification has been accomplished following two principal techniques, known as formal and functional verification [9]. FV is mainly simulation based. Although new methodologies that combine formal, semi-formal and functional solutions have been proposed [10] and adopted by the industry, these methodologies are still limited. In that context, this paper described a tool that enhances the FV traditional flow. Therefore, results comparing different simulators were shown on Fig. 4. Later the layering scheme was detailed and an example was provided as well. Combining the simulation speed of the integrated simulator with the GUI that allows the creation of complex test scenarios, it is possible to perform the FV of designs without writing a single line of code, lowering the verification effort considerably.

4. References