VerTGen: An Automatic Verilog Testbench Generator for Generic Circuits

Shahid Ali Murtza and Osman Hasan
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Islamabad, Pakistan
Email:{smurtza.msee15seecs,osman.hasan}@seecs.nust.edu.pk

Kashif Saghar
Software Quality Assurance Department
Centers of Excellence in Sciences
and Applied Technologies
Islamabad, Pakistan
Email:kashif.saghar@gmail.com

***Abstract*—With the ever growing complexity of hardware designs, their functional verification has become quite a challenge. Despite other techniques like emulation and formal verification methods, simulation continues to be the most common and primary technique to functionally verify the hardware design written in Verilog. Due to the limited computational resources, exhaustive testing of the present-age complex hardware designs is not possible. This makes the selection of vectors very important. However, manual selection of test vectors has often resulted in biased testing and their is always a risk of missing some important corner case. Random testing may solve this problem but manual development of random test generation is a time consuming task. The main contribution of this paper is to facilitate the testbench development task as it presents an automatic random testbench generator tool, VerTGen, for Verilog models. VerTGen has a user friendly GUI and supports all the major probability distributions and can be used to create testbenches for both combinational and sequential circuits. For illustration, the paper presents the testbench creation process of a shift register using VerTGen.**

I. Introduction

Functional verification is the process of checking the logical and sequential properties of a given digital design. With the rapid growth of system design complexities, functional verification of hardware designs has become quite challenging [1]. Functional verification of a digital design is usually carried out by a verification engineer by providing the design environment to the given design by the means of a stimulus or testbench, that can generate input test streams to monitor the design for a given reference output. The verification process can take upto 60% to 70% of the overall design effort [2], [3], and therefore automatic tools that facilitate functional verification are extremely desirable to meet the stringent time-to-market constraints.

Despite the benefits of emulation and formal verification, simulation continues to be the most commonly used in verification method for hardware designs [1]. Most of the hardware models are expressed in a hardware description language (HDL), like Verilog or VHDL. A testbench, which is a series of input vectors, is used for their testing via simulation. One of the main issues of the simulation testbench is to generate the input stimuli that can depict the design environment for revealing all the interesting corner cases [3]. Usually the test streams are generated manually, which is firstly a very time consuming task for large and complex designs and secondly is known to be biased based on the users mind set. The second challenge has been addressed by using random testing along with assertion based verification [4]. The main idea here is that test vectors can be created randomly based on certain probability distribution, like Normal or Exponential, for all the input signals. The desired characteristics of the design can be embedded in the Verilog code as assertions and these assertions can raise a flag if a property fails during the random testing based simulation of the design. Such kind of testing is unbiased and is known to catch corner cases. However, the testbench is still required to be manually generated for every new design.

978-1-5090-3552-6/16/$31.00

To overcome the above-mentioned challenge, in this paper, we propose automatic tool, VerTGen, for creating Verilog testbenches using random vector generation based upon different distribution functions, like Normal, Guassian or Erlang etc., to generate the input test streams. The tool supports a user friendly GUI and facilitates the user to select the desired parameters on the GUI by providing recommendations during the main steps only. To the best of our knowledge, this is the first open source automatic random testbench generation tool.

The rest of the paper is organized as follows: The next section presents an overview of the related work in the domain of testbench and test vector generation techniques. Section III gives details about the proposed VerTGen tool, and in Section IV, a demonstration of how to use VerTGen is explained with an easy to understand example.

II. Related Work

Historically, there have been several methodologies in the literature dealing with functional verification of Verilog/HDL [5], [6], but most of them lack ease of use, generality and automation [1], [7]. Some of the commercially available tools from SynaptiCAD, Xilinx and MATLAB also facilitate in automatic testbench generation for Verilog/VHDL codes. The VeriLogger Extreme [8], a simulator by SynaptiCAD, supports the automatic graphical testbench generation. The

tool helps extracting the top level module ports and gives to the timing diagram of the simulator and lets its users to quickly draw the waveform to depict input stimuli and thus analyze the results. A prerequisite to use the tool is that the user must be familiar with drawing the time waves depending upon the system requirements. The TestBencher Pro [9], i.e., a utility also by SynaptiCAD, can generate testbenches, which create bus functional models from timing diagrams independent of the hardware description language. The technique used in the tool first represents each bus-transaction graphically, and then generates the code for the transaction. This tool also requires the timing diagram to be given as input. The StateBench [10] comes with the StateCAD tool by Xilinx and attempts to aid in automatic testbench generation. The tool automates Verilog/VHDL testbench generation of StateCAD diagrams. The StateCAD [10] requires the state diagram for the model to generate the HDL code, which is further used by StateBench utility. The HDL Verifier [11], by MATLAB also has the ability to automatically generate testbenches for Verilog/VHDL design verification but the tool requires MATLAB or Simulink for cosimulation with HDL simulators. The drawback of the cosimulation based setup is the extra overhead due to interface handshaking and parallel sessions of HDL simulators and MATLAB, as highlighted by Jia [12]. Moreover, the tool also requires the user expertise in working with the cosimulation setup, and MATLAB coding or Simulink environment. Similarly, the hoTBench.net [13], is a web-based application to facilitate testbench development for the design modules in VHDL and Verilog. This tool mainly supports the test pattern modeling and is only commercially available. Despite the fact that these tools aim to facilitate the user in test best generation but every tool has its own limitations in terms of generality and compatibility.

A few open source tools are also available that attempt to generate, or try to assist the generation of the automatic testbench for Verilog based designs. Mishra et. a1. [14] proposed a specification based test generation methodology. However, the technique deals only with the specific problem and lacks the generalization of the methodology. In [7], da Silva et. al. proposed an automatic testbench generator named VeriSC for functional testing of SystemC based environment for verification of hardware designs. The technique is good in vector generation but this is for SystemC based designs. The AutoTestbench [15] is a free plugin for ModelSim simulator to generate Verilog testbench file. It supports generation of the testbench template alongwith component instance generation. However, it lacks the support of Verilog-1995 syntax. The plugin is so basic that it requires a single variable port declaration in the main module, which limits its usability. Moreover, it does not have automatic test vector generation support.

Some efforts to use other languages in the loop for testing Verilog designs have also been made. Usually in this type of methodologies, the HDL is first converted to a host language and then the testbench is accordingly generated and simulated. Such examples can be found in the research works [16]–[18].

III. Proposed Approach

The proposed design for automatic testbench generator is shown in Figure 1. The tool accepts the Verilog design file as input and requires the user to provide the required parameters, i.e., type of the input, range, distribution and seed value in case of default type, and clock frequency in case of the clock signal, total number of vector generation and number of hold clocks, and it returns the generated testbench for corresponding input file in Verilog. The block diagram is composed of three main blocks: File Parsing, Test Vector Generation and Clock and Event Generation, and three other blocks depicted mainly for user inputs via a graphical user interface. These are discussed in detail one by one now.

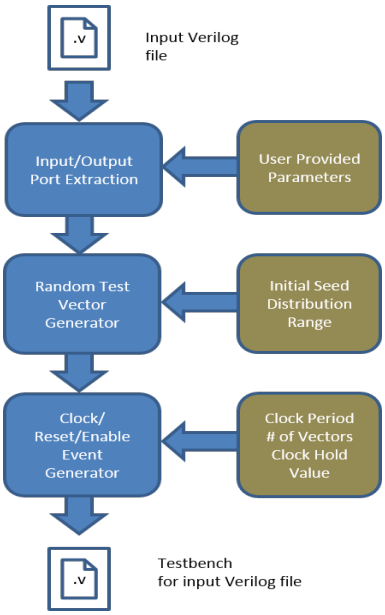

Fig. 1. Proposed Approach for VerTGen

A. Verilog File Parsing

The main purpose of file parsing is to extract the ports (input and output) information of the given Verilog circuit. The file parsing recognizes both Verilog styles namely 1995 and 2001. It extracts direction and size of the ports while ignoring the code comments in the original Verilog file. The extracted ports are then sent to the next block for further processing.

B. Input Parameters: User Interaction 1

Once the inputs are separated from the port list, the tool requires the user to decide the type and constraint on inputs. The type of an input can be either a clock, reset, enable or default, which subsumes all other types apart from clock, reset and enable. The constraints on the default inputs include a range, i.e, minimum and maximum value to be tested and other parameters related to the default inputs include the seed value for randomization and the distribution types.

C. Test Vector Generation

The default type inputs need a simple vector generation for functional testing. Therefore, we adopted the randomization already available in the Verilog language support to handle these inputs. There are quite a few distributions available in Verilog that can be used to randomize the process of vector generation. This helps in automation and generalization of the vector generation for testing. However, a problem with the Verilog random distributions is that true random numbers are not generated as they all depend upon the initial seed value and if this is fixed then the same stream is generated every time. We tackled this issue by generating a new seed value every time randomized vectors are required. We did that by acquiring this seed value from the CPU clock. This feature is not supportable directly from within the Verilog and hence a PLI routine, e.g., in C++ , was required for this purpose. Our implementation also supports acquire the initial seed value from the user besides the option of acquiring it automatically from the CPU time. Moreover, after every run of the testbench, the last value generated by the random stream is saved as the initial seed value for the next run of the test. Once the seed value is selected then any of the built-in distributions, such as Erlang, Exponential, Normal, Uniform and Poisson, could be used for random test generation.

D. Clock and Event Generation

The clock, reset and enable inputs in the Verilog module need special care while generating test streams for them. The clock is a periodic input of square wave of usually fifty percent duty cycle. The clock is thus generated with a user specified frequency. For the reset and enable signals, events are used to trigger the sequence of their assertion and de-assertion.

E. Other Parameters: User Interaction 2

Other parameters that a user has to specify are total number of vector generations and, in case of sequential circuits, user has to tell the number of clocks that an input signal needs to stay unchanged so that the effect of the test stimuli can be propagated to the output for proper functional testing of the Verilog sequential circuit.

F. GUI for the User Facilitation

The testbench generator is a C++ based GUI built on Visual Studio 2013. A snapshot from the GUI is shown in Figure 2. The GUI validates the input and other required parameters during the user interaction phase. The GUI mainly supports the following activities.

- Upload a File: The VerTGen GUI accepts files of Verilog type only from any directory in the system it is running on. Thus, for example, if a testbench file is selected to be uploaded then it gives an error message and redirects to re-upload file.

- Saving the Inputs: Every input needs to be saved by clicking the 'save input' button and this way VerTGen checks if there are any missing parameters and if something is missing then it generates an error message. For the default type inputs, the Distribution and Range can be specified along with the initial seed value, which could be either the value of current time or a user specified value as described above.

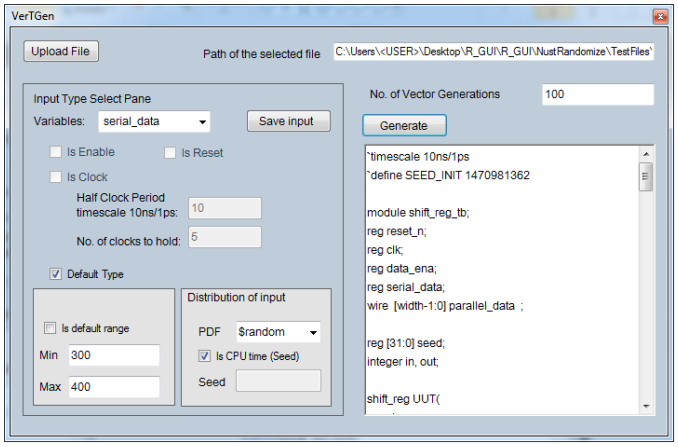

Fig. 2. VerTGen GUI

- Generating the Testbench: VerTGen generates the testbench when the 'Generate' button is clicked. This button upon clicking, first validates if all the inputs are specified, and the number of vector generation value is specified. If everything is OK, then it generates the Verilog testbench file corresponding to the input file.

- Context Menus: There are two context menus in VerTGen: 'Variables' and 'PDF' (probability density function). The Variables menu lists the extracted inputs and allows the user to choose the type of Verilog module inputs and then specify their corresponding parameters. For example, for the clock input, the user has to select the 'clock' checkbox as the type of input and specify half clock period in the corresponding field. The PDF (probability density function) menu provides a list of Verilog system tasks for distributions to be selected by user for default type of inputs.

- Besides the above-mentioned fields/activities, the user also needs to provide the values for the clock frequency, number of clocks to hold for each input, range of default type inputs and the user-defined initial seed of the distribution and total number of vector generations. There are two output text fields: The first one on the upper right corner shows the path of file and the second one the right side shows the generated testbench, which also is saved in the same directory as of the input file by default.

IV. Case Study: Shift Register

In order to demonstrate the working of VerTGen, we show the complete testbench generation flow using the example

of a shift register circuit. The Verilog code that we used for the shift register circuit is given in Code Listing 1, using Verilog-2001. It has one output port, i.e., *parallel_data*, and has four input ports, i.e., *clk, rest_n, data_ena* and *serial_data*.

Listing 1. Verilog main module

```
//------------------------------
// Example: Shift Register with
// parametric width (Verilog-2001)
//------------------------------

module shift_reg #(parameter width=8) (
input reset_n,
input clk,
input data_ena,
input serial_data,
output logic [width-1:0] parallel_data);

always @ (posedge clk, negedge reset_n)
  if(!reset_n)
     parallel_data <= '0;
  else if (data_ena)
     parallel_data <= {serial_data, parallel_data[width
          -1:1]};
endmodule
//------------------------------
```

As explained earlier, the user of VerTGen only needs to provide the file, input types and the simple parameters, like clock frequency, number of vector generation etc., for the testbench generation for the given Verilog code. The following are the simple steps to take for generating the Verilog testbench using our tool.

1) The user first needs to upload a valid Verilog file from any local directory. In the case of the considered Shift register, it would be the file, given in Code Listing 1.

2) The next step is to select the input types via checkbox on VerTGen GUI and their corresponding parameters if any. In the case of the shift register, these inputs would include *clk, rest_n, data_ena* and *serial_data* with types clock, reset, enable and default, respectively.

3) Then, the input needs to be save before going to next input setting. This feature also validates the input value along with its parameters to be compatible; For example, for range of default input, VerTGen basically checks the given value with the given maximum value that the variable can acquit to see if a valid input value was given or not, and vice versa.

4) After all the inputs and their corresponding parameters are set, the next step is to provide a value for total number of vector generations field.

5) The final step is to click on the generate button. This checks if all the inputs and the other parameters are set. If it is OK, then it generates the testbench file for the input Verilog file. The generated file is saved in same directory from where the input file is uploaded. The generated text is also shown on the right text pane of VerTGen GUI.

The parameters provided for the shift register example are as follow:
Input settings:
clock:clock frequency=10, hold clocks=5;
serial_data:PDF=$random, Min=300, Max=400, seed=default;
Others:
Number of vector Generations=100;

The generated testbench is shown in the Code Listing 2. The comments in the code listing provide a brief description of the output code pieces.

Listing 2. Generated Testbench

```
//------------------------------
// Generated Testbench
// Using VERTGEN
//------------------------------

// Time Scale
`timescale 10ns/1ps
// CPU Time for first time initial SEED
`define SEED_INIT 1466922921
// Parametrization value of 'width'
`define width=8;

// Main Testbench Starts here.
module shift_reg_tb;

reg clk;
reg reset_n;
reg data_ena;
reg serial_data;

wire  [width-1:0] parallel_data;

reg [31:0] seed;
integer in, out;

shift_reg UUT(
 reset_n,
 clk,
 data_ena,
 serial_data,
 parallel_data);

// EVENT for reset trigger &  EVENT for reset done trigger
event  reset_n_it;
event   reset_n_done;

// Routine for reset event
initial
 begin
   forever
    begin
        @ (reset_n_it);
            @ (negedge  clk);
               reset_n = 0;  //Assert Reset
            @ (negedge  clk);
               reset_n = 1; //De-Assert Reset
             ->  reset_n_done;
    end
end

// Reset triggering at time unit 10
initial
 begin
            #10-> reset_n_it;
end
```

```
56
57 // Clock Generator
58 initial  clk= 1'b0;
59 always
60     #10  clk=~ clk;
61
62 // Enable Event routine (triggered after reset is done!)
63 initial
64  begin
65   #10
66   @(negedge  clk);
67    data_ena = 0;
68   repeat (5)
69   begin
70    @ (negedge  clk);
71   end
72    data_ena = 1;
73 end
74
75 // Random Input streams generation code piece.
76 integer i_loop;
77 initial begin
78         in  = $fopen("seed.txt","r");
79         // Checks if the file is already created for new
               seed
80         if(in==0)
81            seed='SEED_INIT;
82         else begin
83            r={$fscanf(in," %b\n",seed)};
84            $fclose(in);
85         end
86
87     // Random stream for default input types
88         for( i_loop = 0; i_loop < 100; i_loop = i_loop +1)
                begin
89                #5  serial_data = 300+{$random(seed)}%100;
90         end
91
92     // Saving the last value as new seed for next run
93         out = $fopen("seed.txt","w");
94         $fwrite(out," %b\n",seed);
95         $fclose(out);
96
97 end
98
99 // Output printing for Simulator
100 initial begin #10
101  $display("Simulation Result are as follows:");
102  $monitor(" reset_n: %b,
103             clk: %b,
104             data_ena: %b,
105             serial_data: %b,
106             parallel_data: %b",
107             reset_n,
108             clk,
109             data_ena,
110             serial_data,
111             parallel_data);
112 end
113
114 initial
115  #500 $finish;
116
117 endmodule
118 //-------------------------------------
```

V. Conclusions

In this paper, we presented an automatic Verilog testbench generation tool, VerTGen, that facilitates users in generating randomized testbenches for the functional verification of Verilog circuits. The tool is user friendly and automatic in its working. A user with no or little prior knowledge about Verilog testing can use our tool with ease. VerTGen is available for download at [19] for further usage. It can be observed from the generated testbench code for our illustrative example, given in code Listing 2, that writing such testbenches manually is not a simple job, which clearly shows the practical applicability of the proposed tool. In the future, we plan to integrate VerTGen in the assertion based functional verification process and thus demonstrate the usefulness of VerTGen in identifying bugs in digital designs.

References

[1] J. Bergeron, *Writing testbenches: functional verification of HDL models*. Springer Science & Business Media, 2012.

[2] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip verification: methodology and techniques*. Springer Science & Business Media, 2007.

[3] C. Pixley, A. Chittor, F. Meyer, S. McMaster, and D. Benua, "Functional verification 2003: Technology, tools and methodology,"

[4] A. Groce, G. Holzmann, and R. Joshi, "Randomized differential testing as a prelude to formal verification," in *29th International Conference on Software Engineering (ICSE'07)*, pp. 621–631, IEEE, 2007.

[5] L. Fournier, Y. Arbetman, and M. Levinger, "Functional verification methodology for microprocessors using the genesys test-program generator. application to the x86 microprocessors family," in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, pp. 434–441, IEEE, 1999.

[6] J. Monaco, D. Holloway, and R. Raina, "Functional verification methodology for the powerpc 604 microprocessor," in *Proceedings of the 33rd annual Design Automation Conference*, pp. 319–324, ACM, 1996.

[7] K. R. Da Silva, E. U. Melcher, and G. Araujo, "An automatic testbench generation tool for a systemc functional verification methodology," in *Integrated Circuits and Systems Design, 2004. SBCCI 2004. 17th Symposium on*, pp. 66–70, IEEE, 2004.

[8] SynaptiCAD, "The VeriLogger Extreme webpage," 2016.

[9] SynaptiCAD, "The TestBencher Pro webpage," 2016.

[10] Xilinx, "StateBench and StateCAD webpage," 2016.

[11] MathWorks, "The HDL Verifier webpage," 2016.

[12] J. E. T. Jia, "Reuse MATLAB Functions and Simulink Models in UVM Environments with Automatic SystemVerilog DPI Component Generation (Article)," 2016.

[13] HoTBench.net, "The hoTBench.net webpage," 2016.

[14] P. Mishra and N. Dutt, "Graph-based functional test program generation for pipelined processors," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 1, pp. 182–187, IEEE, 2004.

[15] AutoTestbench, "AutoTestbench webpage," 2016.

[16] M. Westmeier, B. Herwig, and J. Börcsök, "Enhancing a simulation environment for computer architecture to a systemc based testbench tool for design verification," in *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, pp. 1–6, IEEE, 2011.

[17] D. Becker, R. Singh, and S. Tell, "Software/hardware co-simulation with verilog and c++," in *Proceedings 1st International Verilog HDL Conference*, pp. 33–37, 1992.

[18] W. Snyder, "Verilator and systemperl," in *North American SystemC Users' Group, Design Automation Conference*, 2004.

[19] S. A. Murtza, "VerTGen webpage," 2016.