

A Package for Test Hardware Evaluation

Nastaran Nemati, Majid Namaki, Zahra Najafi, Zainalabedin Navabi

*Electrical and Computer Engineering Department
Faculty of Engineering – Campus #2 – University of Tehran, 14399 Tehran, IRAN
{nastaran, mnamaki, znajafi, navabi}@cad.ut.ac.ir*

Abstract

In this work, a test hardware evaluation package based on Verilog and its procedural language interface is presented. This package contains components like test insertion, fault simulation, etc, and its focus is on DFT and BIST evaluation and determining their necessity.

I. INTRODUCTION

Test and Testability are important issues in digital hardware design. Due to the ever increasing complexity of digital systems, the required time for testing these designs is becoming more critical. Usually the test process takes about 30 to 40 percent of the manufacturing costs. On the other hand, due to the necessary short time-to-market, testing must be considered from the very first phases of the design process. In recent decades, including some extra hardware around the original design to make it testable has become a common practice. This concept is referred to as Design for Testability or DFT, which means inserting some test hardware into the original design to test it or increase its testability. Test hardware insertion into DUT can most effectively be done at the RT level. In this work, a package to decide on the necessity of test hardware insertion and its evaluation is proposed. Since hardware description languages (HDLs) are the most common RTL environments, this package is developed based on Verilog and its procedural language interface (PLI). The interface and Verilog testbench capabilities provide a designer-friendly work environment.

II. TEST HARDWARE EVALUATION PACKAGE

A. HDL/PLI Environment

PLI provides a library of C language functions that can directly access data within an instantiated Verilog HDL data structure. With PLI, the advantages of doing testable hardware design in an HDL, and having software tools for manipulation and evaluation of designs can be achieved at the same time. Therefore, not only the design core and its

testbench can be developed in a uniform programming environment, but also all the facilities of software programming (such as complex data structures and utilization of functions) are available. So, test methods can be performed in such a mixed environment more easily and without having to mingle with the original design. Using PLI, a designer can develop a virtual tester for test hardware inserted in the DUT. This virtual tester can be used for evaluation of various DFT implementations. This requires no modifications in the HDL code of the hardware under test and its test hardware.

B. Test Hardware Development and Evaluation

Several PLI utilities that are useful for developing proper test hardware into the DUT are provided in our HDL toolbox. Utilities include functions for fault injection, fault removal, fault collapsing, controllability and observability measurements and signal activity estimator. Some of these utilities are useful to detect when and where incorporating test hardware into the original design is necessary and to find the best parts of the design to insert this extra hardware. Other utilities are for evaluation of the resulted test hardware. However, the information obtained from the evaluation part can be fed back into the diagnosis and development (D&D) part and help developing better test hardware.

Figure 1 shows various parts of this environment. To provide the required tools for the D&D and evaluation parts, the PLI utilities mentioned above are used within HDL testbenches for developing test applications such as test point insertion, serial fault simulation, test generation algorithms, hierarchical fault simulation, semi-deterministic test generation, BIST evaluation, and DFT evaluation. The most important feature of these test applications is that they are provided in a parametric form. In other words, the required testbenches for developing these test applications for various DUTs can be provided in a semi-automatic way just by setting a number of parameters in the template testbenches.

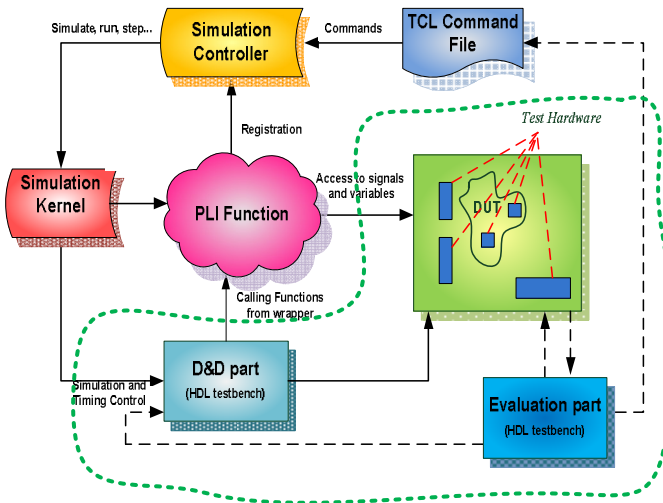


Figure 1. Environment of Test Hardware Evaluation Package

C. A DFT Evaluation Example

As mentioned, the test applications in this package are HDL testbenches invoking PLI utilities. For example, Figure 2 shows the pseudo code of a parametric Verilog HDL testbench that performs DFT evaluation. At the very beginning of this code, faults that might occur in the DUT with DFT inserted in it are dumped into a fault list file. Then each fault is inserted and appropriate test data is shifted into the inserted scan chain. Along with this shift in phase, the current states of the DUT are also shifted out. By comparing the collected primary outputs and shifted out states with expected ones, the fault coverage and test time for the inserted DFT can be achieved. Afterwards, this information together with the information obtained from the diagnosis and development part can be used to reconfigure the DFT and develop a more reliable one with less overhead.

```

module DFT_VirtualTester();
DUT_ScanInserted inst(clk, rst, in_vec, out_vec, en, NbarT, Si, So);
Initial begin
  $FaultCollapsing ( DFT_VirtualTester.inst,
"DUT_ScanInserted.flr");
while( !feof(faultFile) ) begin
  $fscanf(faultFile, "%s s@%b\n", wireName, stuckAtVal);
  numOfFaults = numOfFaults + 1;
  $InjectFault ( wireName, stuckAtVal );
  detected = 1'b0;
  @(posedge clk);
  for( i = 0; ( i < numOfTVs ) && ( !detected ); i = i + 1 ) begin
    //Sample Primary Outputs
    sampledPOs <= out_vec;
    line <= testMem[i];
    //Shift out previous state, simultaneously shift in new state
    @(negedge clk);
    NbarT <= 1'b0;
    for( j = 0; j < numOfDffs; j = j + 1 ) begin
      Si = line[j]; lastState[j] = So;
      @( negedge clk );
    end
    //Apply PIs
    NbarT <= 1'b0;
    in_vec <= line[numOfPIs + numOfDffs - 1 : numOfDffs];
    @(posedge clk);
    //Check Previous POs and Previous State
    if( { sampledPOs, lastState } !=
line[ numOfPIs + numOfPOs + 2 * numOfDffs - 1 :
numOfPIs + numOfPOs + numOfDffs - numOfPOs] ) begin
      detected = 1'b1;
      numOfDetected = numOfDetected + 1;
    end
  end// for
  $RemoveFault ( wireName );
end// while
//Calculate Coverage
always #10 clk = !clk;
endmodule

```

Figure 2. Pseudo code for DFT Evaluation