Utilizing HDL Simulation Engines for Accelerating Design and Test Processes

Najmeh Farajipour, S. Behdad Hosseini and Zainalabedin Navabi
Electrical and Computer Engineering Department
Faculty of Engineering – Campus #2 – University of Tehran
14399 Tehran, IRAN
{najmehf, behdadh, navabi}@cad.ece.ut.ac.ir

Abstract—This paper introduces a complete test package in VHDL that makes it possible to fault simulate and test generate a component during its design process. Different approaches on test applications can be combined and then be applied to combinational, sequential and scan-based circuits in a fully configurable and convenient environment. For demonstrating the power of VHDL in test configurations, we used two different approaches to fault simulation and combined them with a random test generation.

Keywords: Digital Test, Fault Simulation, Fault Collapsing, VHDL, Simulation

I. INTRODUCTION

Hardware description languages are used extensively in describing a complete digital system with different levels of abstraction [1]. In a standard design procedure, fault simulation and generating an appropriate test set is carried out after the design phase has been completed. It is possible to merge design and test generation phases in a complete chip manufacturing process utilizing the rich semantics [1] available in VHDL. In other previous works like [8] and [9], VHDL has been used as an assistant tool for describing fault models and in a limited manner.

In our proposed framework a collection of fully reconfigurable VHDL models are used. The primitive gate models can be bound using configuration specifications [1] to an appropriate gate model which can be the same used for simulation. Also different methods for fault collapsing, fault simulation and test generation can be implemented and used in any desired combination and applied to any supported circuit including combinational, sequential and full-scan based models simply by changing global constants and using different configuration specifications.

The main advantage of our framework is that the same EDA tool used for simulating the designs are also used as fault simulator and test generator tool. Therefore the designer works with a single tool and the overall time needed for design and test generation can be reduced. Also, the time needed for executing a fault simulator and test generator on a complete design can be reduced considerably by applying them early to a single component in a big design [8].

The timing characteristics of the circuit, to be used for instance in detecting delay faults [3], is applied without any change to fault simulator and test generator. Therefore when using a different library or another manufacturing technology, all of changes in timings like propagation delays, setup times, etc are applied automatically to our model.

Our proposed framework which is a collection of VHDL models has the ability to be embedded in a conventional simulator without sacrificing its flexibility and generality. It will boost the performance of test application considerably and the need for additional special software for test application can be eliminated.

The rest of the paper is organized as follows: our fault enabled gate and logic models are presented in section II. Fault list generation and a fault collapsing method are explained in section III. Fault simulation of the circuit using our fault model with different approaches is discussed in section IV. Experimental results are presented in section V. Finally, conclusions and summary are drawn in section VI.

II. FAULT-ENABLED GATE AND LOGIC MODELS

We begin our discussion by introducing a fault enabled gate model in VHDL. By reconfiguring the models, designer can switch from one technology to another, and all behavioral and timing considerations are automatically applied to the entire test package.

Our gate models are actually wrappers on original gates that can be instructed to behave faulty. Firstly each gate introduces itself by means of the *GatelistInsert* procedure into a global list of gates. Our gate list is a vector of records where each record completely describes a gate by its path name, type of gate (e.g. a NAND gate), number of inputs, and number of outputs. Attribute *PATH_NAME* is used to distinguish between instantiated gates.

A global signal *activated_fault* is used to indicate which fault is currently active (explained later). Then each gate waits until the current fault or its inputs/outputs are changes. In such situations the gate wakes up and reevaluates the fault-enabled output. If a fault on one of the inputs is injected, virtual input

is made by infecting the input vector according to the fault using *FaultyInput* function. Then this virtual input is passed to the original gate and the virtual output is calculated according to the gate functionality. *FaultyOutput* checks if a fault must be injected on output and changes virtual output appropriately. Also any necessary processing to other fields of output signal (of type *test_ulogic*) is done in this function.

Listing 1 shows the VHDL code for TestGate_N_1 which is the most common model for describing a gate with arbitrary number of inputs but with one output such as an AND gate.

These models must be configured by means of some configuration specifications to a desired library (such as standard 74 series) by binding *OriginalGate* components to corresponding gate. As an example, Listing 2 shows how a fault-enabled AND gate can be made using a simple AND model. Henceforth this *configured_and* can be used instead of AND gates in the gate-level description of CUT.

For maximizing the generality while using the power of signal semantics in VHDL, we have developed a customized logic system on top of *std_logic* named *test_logic*. We have implemented *test_logic_1164* and *test_ulogic_1164* packages functionally the same as their corresponding packages but with test-aware features such as the faulty value of the signal.

Each signal of type *test_logic* has a good value which is the same as *std_logic* and also a collection of fields used in our test algorithms. For example a faulty value is one of the fields of this type which is responsible to carry the faulty value of the current activated fault in our concurrent fault simulation method. Listing 3 shows the *test_ulogic* type and its AND operator implementation.

III. FAULT LIST GENERATION AND COLLAPSING

Before fault simulation, it is necessary to make a list of faults. In this section, we introduce the necessary models in VHDL to generate a complete fault list of the design and also a procedure for a line-oriented fault collapsing.

```
ENTITY TestGate_1_N IS
    GENERIC (tplh, tphl: TIME);
    PORT (ii: IN test_ulogic; oo: OUT test_ulogic_vector);
END ENTITY;

ARCHITECTURE unconfigured OF TestGate_1_N IS
    CONSTANT path: STRING := TestGate_1_N'PATH_NAME;
    SIGNAL virt_ii: std_ulogic;
    SIGNAL virt_oo: std_ulogic_vector(0 TO oo'LENGTH - 1);
BEGIN
    -- Identify itself in the list of gates
    GatelistInsert(path, g, 1, oo'LENGTH);

    -- Virtual (possibly faulty) input value of gate
    PROCESS (ii, activated_fault) BEGIN
        virt_ii <= FaultyInput(path, ii);
    END PROCESS;

    -- Use the original gate for getting virtual output
    U: OriginalGate_1_N GENERIC MAP (tplh, tphl)
        PORT MAP (virt_ii, virt_oo);

    -- (Possibly faulty) output value of gate
    PROCESS (virt_oo, activated_fault, ii) BEGIN
        oo <= FaultyOutput(path, virt_oo, ii);
    END PROCESS;

END ARCHITECTURE;
```

Listing 1. Fault-enabled gate model in VHDL

```
----------------------- AND_GATE -----------------------
ENTITY AND_GATE IS
    GENERIC (tplh, tphl: TIME);
    PORT (inp: IN std_ulogic_vector; outp: OUT std_ulogic);
END ENTITY;

ARCHITECTURE structural OF AND_GATE IS
    SIGNAL output: std_ulogic;
BEGIN
    -- and of all inputs
    output <= GateVector(inp, G_AND);

    outp <= output AFTER tplh WHEN output = '1' ELSE
            output AFTER tphl;
END ARCHITECTURE;

-------------------- configured_and --------------------
CONFIGURATION configured_and OF TestGate_N_1 IS
    FOR unconfigured
        FOR ALL : OriginalGate_N_1
            USE ENTITY Test.AND_GATE (structural)
            GENERIC MAP (tplh, tphl)
            PORT MAP (inp, outp);
        END FOR;
    END FOR;
END CONFIGURATION;
```

Listing 2. A fault-enabled AND gate

Like the gate list, our fault list is also a shared variable vector of records; each record specifies a possible fault in the circuit. Each fault record consists of a faulty gate (an index into gate list), a tag to show whether the fault is on inputs or outputs, line number of fault on inputs or inputs and finally the stuck-at value.

Two methods for fault generation are implemented. The first one generates all possible stuck-at faults in the circuit. SA-0 and SA-1 fault on each input and output of every gate is inserted into the fault list in this method.

Second fault generation method is a line-oriented fault collapsing [2], which only inserts a representation of a set of equivalent faults on the list. The method simply inserts faults only on inputs of gates according to the Table 1.

IV. FAULT SIMULATION

A fault simulation can be executed using a previously build fault list by the same simulation engine used for normal simulation of the design. These models are entities and architectures that connect to the model under test and drive its primary inputs and observe the primary outputs.

```
TYPE test_ulogic IS RECORD
    -- Original value of the signal
    good: std_ulogic;

    -- Added fields to the signal
    faulty: std_ulogic;
    ptr_index: NATURAL;
    line_level: POSITIVE;
    ...

END RECORD;

TYPE test_ulogic_vector IS ARRAY (NATURAL RANGE <>) OF
    test_ulogic;

FUNCTION "and"  ( l : test_ulogic; r : test_ulogic )
    RETURN test_ulogic IS
    VARIABLE v: test_ulogic
BEGIN
    v.good := l.good AND r.good;
    ...

    RETURN v;
END FUNCTION;
```

Listing 3. Fault-enabled logic model in VHDL

Table 1. Line oriented fault collapsing

TYPE OF GATE	FAULT(S) ON INPUTS
AND, NAND	sa1
OR, NOR	sa0
INV, BUF, PI	None
FANOUT, XOR, XNOR, DFF, PO	sa0, sa1

We have developed two fault simulation methods. The first one is the conventional serial fault simulation. Improvements in time of fault simulation algorithms can be reached in two ways: minimizing the evaluation cost of each gate, and minimizing the number of faulty circuit calculations [5]. A novel approach by combining concurrent fault simulation and path-sensitizing is proposed based on the second improvement idea.

A. *Serial Fault Simulation*

Serial fault simulation [3] is the simplest approach for fault simulation. For a given input test pattern, it works by injecting faults from undetected (remaining) fault list one by one and simulates the circuit and compares the output to golden response of the circuit. If those responses differ in at least one bit, the fault is detected and otherwise not. In our framework, input test pattern can be read from a file or feed from a higher level architecture like a test pattern generator.

The algorithm is adapted for our proposed gate models by simply changing the global signal *activated_fault*. The id of the desired fault is assigned to this signal and faulty response of the circuit will be obtained from the output after propagation of faulty signals.

B. *Concurrent Path-sensitizing (CPS) Fault Simulation*

In this approach, we have combined concurrent fault simulation [7] and path sensitizing method [6] for fault simulation by utilizing the best parts of each one. Therefore a simple and fast fault simulation algorithm can be obtained.

Assume that a test pattern has been applied to primary inputs of the circuit and we tend to detect all detectable faults as fast as possible. In our test environment each line has its own local fault list which includes all faults that is reached to it. First we inject a fault from the remaining fault list to the corresponding line and insert its ID to its local fault list. Then we try to propagate it to a primary output (sensitize a path).

When a fault reaches to an input of a gate, it can propagate by it to its outputs, which means an insertion is being made to the local fault list of output if good values of other inputs permit it. By permission we mean that the good value of the gate output is different from its faulty value. Thus it can be deduced that the test pattern can detect the corresponding fault on the gate's output. We continue to merge input fault lists and also inserting the current gate faults (faults on its input lines) and put it on local fault list of output of the gate. This process continues until we make the fault lists of the primary outputs. All faults in the fault list of the primary outputs are marked as detected faults by the applied test pattern and will be dropped from the global fault list.

For example consider the circuit in Figure 11. The symbol x/y means the faulty value and good value of the line is 'x' and 'y', respectively. Also A-sa1 means stuck at one fault is on the line 'A' and so on. We have applied "010" to the primary inputs (A,B,C) and injected A-sa1. A-sa1 is inserted to 'A' local fault list. Since the other input of AND gate is set to '1', this fault propagates to the output of AND gate and changes its faulty value to '1'. Therefore D-sa1 will be inserted into the local fault list of 'D' and merged with local fault list of 'A' (since local fault list of 'B' is empty). If the faulty value of 'B' was not '1', then the local fault list of 'D' should be empty. We continue this procedure until we reach the primary output 'E'. All faults in the local fault list of 'E' are detected by the test pattern.

Note that if two or more input line of the gate have faults in their local fault lists we will merge them with each other and output local fault list, if the output faulty value is different from its good value. Otherwise the output local fault list will be empty. Also note that when input fault lists are merged, the memory occupied by them is released for minimizing memory consumption.

This method is very simple to implement using our customized logic system. A signal of type *test_logic* has a field *ptr_index* which points to a shared vector variable of pointers to local fault list. When the input signals of a gate change, the gate process its input lists and will produce a list on output signal as described before.

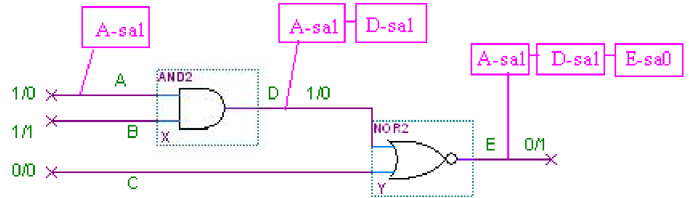

Figure 1. Demonstration of CPS fault simulation

V. EXPERIMENTAL RESULT

We have simulated our model with a simple accompanying random test generation on a Pentium Celeron 2.0GHz with 2GB RAM.

Our test generation method is a pure random one which uses the uniform distribution for generating bits of a test pattern. The method iteratively generates a random test pattern, and invokes one of the previous fault simulation engines. All of detected faults are removed from remaining faults when moving to the next step. The method continues until desired fault coverage is reached or no advancement is observed after a predefined number of tries.

Our results show that a considerable improvement in simulation time is obtained when switching to CPS fault simulation.

In our proposed CPS method, the worst case happens when only one fault (the selected fault) is detected in each iteration and because we don't have to search in a state-space nor

backtracking, the time complexity of an iteration of our algorithm is not worse than the time complexity of an iteration of serial fault simulation. Also we propagate faults (instead of signals) in our model just like in serial fault simulation with the aid of VHDL simulator. Our method outperforms the two other ones, because more than one fault can be detected in each iteration.

VI. CONCLUSION AND FUTURE WORK

We have presented a complete, convenient and easy to use environment for test applications in VHDL that enables a designer to simultaneously investigate fault simulation and test pattern generation for her model while other parts of the design is still described behaviorally.

Also two different methods for fault simulation with attention to the underlying event-driven mechanism of current conventional simulations have been proposed and the effect of combination of each one with a random test generation procedure was presented.

We have shown that by using a customized fault-enabled gate and logic models, it is possible to develop all test applications in VHDL while having the flexibility of any desired combination of available methods and also different circuit types.

We plan to embed our fault-enabled gate and logic models into an event-driven VHDL simulator in our future works. It will give us the possibility to obtain the maximum speedup available for test application in VHDL to make their speed comparable to their software counterpart written in low level languages such as C. Also we like to develop more fault simulation methods and pseudo random test generation algorithms to our package and observe the behavior of different combinations of them in simulation time and fault coverage.

REFERENCES

[1] Z. Navabi, "VHDL: Modular Design and Synthesis of Cores and Systems," *McGraw Hill,* 1998.

[2] M. Nadjarbashi, Z. Navabi and M. R. Movahedin, "Line Oriented Structural Equivalence Fault Collapsing," in *IEEE Workshop on Model and Test,* 2000.

[3] A. Miczo, "Digital Logic Testing and Simulation," *John Wiley & Sons,* Hoboken, New Jersey, 2003.

[4] L. Wu and D. M. H. Walker, "A Fast Algorithm for Critical Path Tracing in VLSI Digital Circuits," in *IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems,* 2005.

[5] S. Mirkhani and Z. Navabi, "Enhancing Fault Simulation Performance by Dynamic Fault Clustering," in *Proc. of the 14th Asian Test Symposium,* 2005.

[6] J. Silva and K. Sakallah, "An analysis of path sensitization criteria," in *Proc. Int. Conf. on Computer Design,* 1993.

[7] E. G. Ulrich and T. G. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," in *IEEE Trans. On Computers*, Vol. 7, No. 4, 1974.

[8] Z. Navabi, S. Mirkhani, M. Lavasani, and F. Lombardi, "Using RT-Level Component Descriptions for Single Stuck-at Hierarchical Fault Simulation," *Journal of Testing: Theory and Applications (JETTA),* Vol. 20, No. 6, December 2004.

[9] Z. Navabi and M. Shadfar, "A VHDL Based Test Environment Including Models for Equivalence Fault Collapsing", *Proc. of VHDL International Users' Forum*, 1994.