

# **Hierarchical Test Generation for VHDL Behavioral Models**

**Sanat R. Rao   Bi-Yu Pan<sup>1</sup>   James R. Armstrong**

The Bradley Department of Electrical Engineering  
Virginia Polytechnic Institute and State University  
Blacksburg, VA 24061

E-mail: rao@vtsda.sda.vt.edu

Ph: (703) 231 - 4202

## **Abstract**

In this talk, a novel approach to test generation for VHDL behavioral models is described. An algorithm called HBTG, Hierarchical Behavioral Test Generator, has been developed and implemented to systematically generate tests for VHDL behavioral models. HBTG accepts the Process Model Graph and the pre-computed tests for the individual processes of the model, from which it hierarchically constructs a test sequence that exercises the model. The construction of the test sequence is automatic provided that the tests for individual processes of the model are available. The test sequence derived is used for simulation of the model. The modeler is thus relieved of the time-consuming problem of developing test benches.

<sup>1</sup>Bi-Yu Pan is currently with Intel Corporation, Folsom, CA.

## **HIERARCHICAL TEST GENERATION FOR VHDL BEHAVIORAL MODELS**

October 21, 1992

Sanat R. Rao

Bi-Yu Pan

James R. Armstrong

The Bradley Department of Electrical Engineering  
Virginia Tech  
Blacksburg, VA 24061

E-mail: rao@vtsda.sda.vt.edu

Ph: (703) 231 - 4202

### **OUTLINE**

- Test Generation for VHDL Behavioral Models
- Concept of Hierarchical Test Generation
- The Modeler's Assistant
- Test Generation Approach
- The HBTG Algorithm
- Test Generation Example
- Coverage Measure
- Complete CAD System

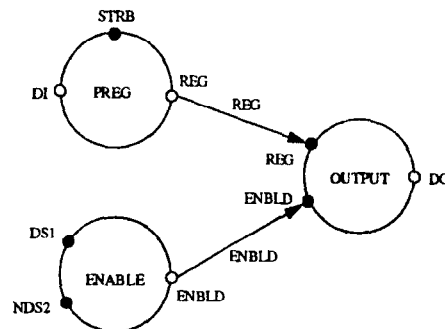
## TEST GENERATION FOR VHDL BEHAVIORAL MODELS

- Once the VHDL behavioral model of a circuit is developed, test patterns are generated and applied for simulation of the model.
- By observing the simulation output, the functionality of the model can be checked.
- Traditionally, test development for VHDL models has relied on a design engineer or a model developer to generate tests.
  - Time-consuming and labor-intensive process
  - Test sets satisfy no formal definition of completeness.
- The Hierarchical Behavioral Test Generation (HBTG) Algorithm systematically generates tests for VHDL behavioral models.

## CONCEPT OF HIERARCHICAL TEST GENERATION

- The circuit to be tested is broken up into a number of functional modules. Tests for individual modules are precomputed and stored in the design library. These tests are used to compute the test sequence for the entire unit.

### Process Model Graph of the 8-Bit Register



- The HBTG algorithm exploits the hierarchy inherent in a Process Model Graph.
- Given the PMG of the VHDL behavioral model and the primitive test data files for each process of the PMG, the HBTG algorithm generates a test for the whole entity.
- The PMG is created using the Modeler's Assistant.

### THE MODELER'S ASSISTANT

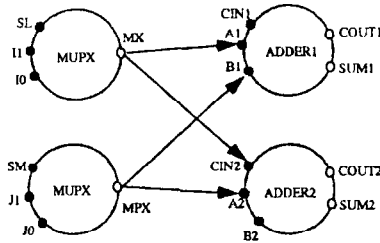
- Graphical CAD tool developed at Virginia Tech.
- User graphically enters the Process Model Graph of the VHDL model to be developed along with the functionality of each process.
- The Modeler's Assistant automatically generates the VHDL behavioral model from the above information.
- A Process Model Graph (PMG) database stores information about the geometry of the PMG.
- The HBTG algorithm extracts information from this database for use during the test generation process.

### TEST GENERATION APPROACH

- In a fault-oriented test generation approach, a fault model is adopted and a test pattern is generated to detect the fault.
- In our approach, no specific fault model is adopted.
- The HBTG algorithm constructs a test sequence which exercises the model thoroughly.
- A process in the architectural body of a VHDL behavioral model will be executed only if there is an event on one of its sensitive input ports.
- Major criterion used by the HBTG algorithm:  
The test sequence should activate as many sensitive ports of the model as possible.
- An **effective test sequence** for a VHDL behavioral model is one which activates all the sensitive ports of the model at least once.

## CONSTRUCTION OF SENSITIVE PATHS

A Sensitive Path is a directed path that starts at a sensitive primary input port (PI) and ends at a primary output port (PO) with the intermediate ports along the path consisting of as many sensitive ports as possible.



Sensitive Path 0:

sp[0][0] = 9(B2)  
sp[0][1] = 5(SUM2)

Sensitive Path 1:

sp[1][0] = 16(CIN1)  
sp[1][1] = 13(SUM1)

Sensitive Path 2:

sp[2][0] = 24(SM)  
sp[2][1] = 21(MPX)  
sp[2][2] = 10(A2)  
sp[2][3] = 7(COUT2)

Sensitive Path 3:

sp[3][0] = 25(J0)  
sp[3][1] = 21(MPX)  
sp[3][2] = 17(B1)  
sp[3][3] = 15(COUT1)

Sensitive Path 4:

sp[4][0] = 26(J1)  
sp[4][1] = 21(MPX)  
sp[4][2] = 17(B1)  
sp[4][3] = 13(SUM1)

Sensitive Path 5:

sp[5][0] = 32(SL)  
sp[5][1] = 28(MX)  
sp[5][2] = 8(CIN2)  
sp[5][3] = 5(SUM2)

Sensitive Path 6:

sp[6][0] = 33(I0)  
sp[6][1] = 29(MX)  
sp[6][2] = 18(A1)  
sp[6][3] = 15(COUT1)

Sensitive Path 7:

sp[7][0] = 34(I1)  
sp[7][1] = 29(MX)  
sp[7][2] = 18(A1)  
sp[7][3] = 13(SUM1)

## HBTG PROCESS

The Hierarchical Behavioral Test Generation (HBTG) algorithm is implemented in C and is currently running on a SUN SPARCstation 2. The algorithm interfaces with the Modeler's Assistant to receive the Process Model Graph of the VHDL model.

The Test Generation process proceeds as follows:

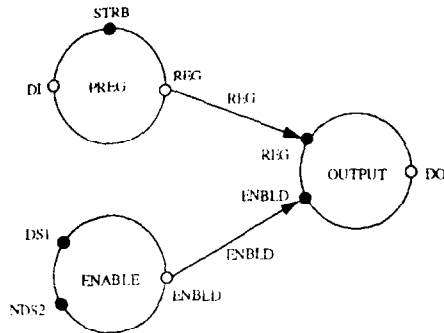
- Sensitive Paths are constructed through the PMG
- Sensitive Paths are selected and activated one by one. The activation of a path starts with activating the first port on the path.
- The signal value is then propagated to the primary output on the path.
- Justification is then done to justify the internal signal values specified during the forward activation process towards the primary inputs.

```

program HBTG
begin
do Construct_Sensitive_Path;
while there is a sensitive path in the PMG not activated, do
  Select the path;
  Decide put;
  Activate the first port along the path;
  Increase the portact of the port;
  while Primary output of the path is not reached, do
    Decide put;
    If the front signal on the path is an event, then
      If the signal is a sensitive signal, then
        do propagation of the event;
        Increase the portact of the port;
      else
        do propagation of the non-sensitive signal;
        Increase the portact of the port;
      end if;
      do implication;
    else
      do propagation of a constant signal value;
      do implication;
    end if;
  end while;
  while a process with known output (Os) and unknown input exists, do
    Select a sensitive path that traverses Os if possible;
    while the primary input on the path is not reached, do
      decide put;
      do signal justification towards primary input;
      do implication;
      Increase the portact of the port;
    end while;
  end while;
end while;
end program HBTG;
  
```

### THE HBTG ALGORITHM

### TEST GENERATION EXAMPLE



Sensitive Paths:

- path 0: NDS2 => ENBLD => DO
- path 1: DS1 => ENBLD => DO
- path 2: STRB => REG => DO

frame	DO	ENBLD	NDS2	DS1	REG	DI	STRB
0	X	X	X	X	X	X	X
2	X	X	X	X	D1	D1	R
1	D1	R	F	1	D1	D1	1
3	Z	F	0	F	D1	D1	1
5	D1	R	0	R	D1	D1	1
4	D2	1	0	1	D1	D2	1

Test Sequence for the 8-bit Register Model

## COVERAGE MEASURE

Copyright (c) 1990 by Synopsys, Inc.

ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys, Inc. and may be used and disclosed only as authorized in a license agreement controlling such use and disclosure.

data date: Tue Jun 2 16:35:20 1992

simulation time: 24

coverage data: reg8.cov

VHDL source: reg8.vhd

Line	Count	Text
<hr/>		
1		
2		use WORK.VHDLCAD.all, work.all;
3		entity reg8 is
4		port (DO: out MVL_VECTOR(0 to 7);
5		NDS2: in BIT;
6		DS1: in BIT;
7		DI: in BIT_VECTOR(0 to 7);
8		STRB: in BIT
9		);
10		end reg8;
11		
12		architecture BEHAVIOR of reg8 is
13		
14		signal ENBLD: BIT;
15		signal REG: BIT_VECTOR(0 to 7);
16		begin

17		OUTPUT_4: process (ENBLD,REG)
18		begin
19		if (ENBLD = '1') then
20	6	DO <= BV_TO_MVL(REG);
21	3	else
22	3	DO <= "ZZZZZZZ";
23	3	end if;
24	6	end process OUTPUT_4;
25		
26		ENABLE_9: process (NDS2,DS1)
27		begin
28		ENBLD <= DS1 and not NDS2;
29	7	end process ENABLE_9;
30		
31		PREG_14: process (STRB)
32		begin
33		if (STRB = '1') then
34	4	REG <= DI;
35	2	end if;
36	4	end process PREG_14;
37		
38		end BEHAVIOR;
39		
40		
41		
42		
43		
44		

### COMPLETE CAD SYSTEM

