

# VHDL Simulation Methodology and Techniques

Robert Yockey  
Paul Handly

**MOTOROLA** Inc.  
Government Systems and Technology Group  
8201 E. McDowell Road  
Scottsdale, Arizona 85252  
yockey@email.mot.com handly@email.mot.com

Secure Systems and Technology Section  
Secure Systems Office

## Abstract

This paper presents a design methodology and interactive test bench techniques for VHDL environments focused on the reduction in the gate level simulation to behavioral description verification cycle time. The methodology leads to the creation of a VHDL test bench simulation environment that simultaneously stimulates both the behavioral VHDL system description and the gate level model so that simulation run-time verification of the outputs can be made. The interactive test bench techniques presented allow the designers at both the system and the component level to streamline test benches and to create a more efficient debugging and testing environment. A detailed example of the methodology and the test bench techniques is presented for the implementation of an IEEE 1149.1 JTAG TAP controller.

## Section 1. Introduction

In the course of engineering design there has always been the fundamental problem of measuring how well the final product matches the design's original intent. Specifically, in modern ASIC development

the original intent of the system designer is often expressed as a requirements specification. The ASIC designer's task is to verify his gate level description against this specification.

This verification often becomes a time consuming and error prone process. This is primarily due to the disjoint nature between the specification and the gate level environments. This disjoint nature can be viewed as a "verification chasm". Ambiguous textual specifications create the worst case chasms to cross. Even concise VHDL model specifications may require much manual effort to cross the chasm.

There are three areas that must be addressed to eliminate the ASIC "verification chasm". First, concise VHDL model requirements specifications are needed. Second, interactive test benches for high utility automated functional verification must be developed. Third, a methodology that includes mixed mode simulation environments must be employed. The result is an automated comparison between circuit specification and circuit realization yielding increased accuracy and reduced cycle time.

This paper will expand on these three areas developing them with VHDL code, figures, and detailed descriptions. These techniques and methodology are presently being used and have proven themselves on **MOTOROLA** GSTG programs.

## **Section 2. Why Behavioral Simulation?**

VHDL provides engineers with an increasingly popular "standard" language for modeling hardware. These models can serve as the requirement specification for complete systems and components within the system. The key advantages to having a VHDL model over textual requirement descriptions are the ability to simulate the specification and the reduction in ambiguity. Once LSI level behavioral simulations are complete, the LSI designers proceed to schematic entry/logic synthesis while system designers immediately begin module level simulation. This provides module verification early in the design cycle reducing unknown risks, allowing time to incorporate changes in the early "less expensive" phases of the design cycle.

The authors experience with the new methodology showed that as a VHDL model was completed and the chip advanced to the detailed design phase, other chips were able to take advantage of the availability of the model as a design and verification tool. Questions about interaction between chips were not postponed until after one of the chips had already gone to mask. Generation of simulation vectors for the module (an MCM in this case) was finished very early.

Old impediments to behavioral simulation included the fact that the vectors created had to be transformed for use at the gate level. The input stimulus would be translated with some sort of translation program or, at worst, manually re-created with the gate level simulator input stimulus generation tools. The output of the gate level simulation had to be visually compared by the designer for verification. Behavioral test benches can rely on unit

under test output to create input vectors that are "interactive" in nature. Most gate level input stimulus generation tools generate the independent vectors without benefit of unit under test output. Often, the disconnect between gate level and behavioral simulation was viewed as being so great that the efforts were considered to be redundant, therefore behavioral simulation was dropped.

## **Section 3. Behavioral Simulation Techniques**

The approach used by the authors is a substantial improvement on old methods. The approach is based on the concept of moving verification to the earliest phase of the design cycle. The design is originally created and verified using behavioral or RTL level VHDL simulation. The design is built in hierarchical blocks so that it can be created and tested at gate and behavioral levels in parallel by different people. The behavioral test bench is interactive, dynamic, and segmented. This new style of test bench allows the designers at both the system and component levels to create a more efficient debugging and testing environment.

An interactive test bench is one that uses outputs from the UUT (Unit Under Test) to determine its behavior. A simple example is a test bench that after applying a stimulus to the UUT, waits for a UUT output event to occur before proceeding to the next input stimulus. A more complicated example of an interactive test bench is one that monitors UUT output in order to compare it with a correct value. The test bench can then automatically alert the user of correct or incorrect performance of the UUT. Interactive test benches can also be used to determine when it is okay to enable tristated buses, if the bus is under control of the UUT. A last example of the utility of an interactive test bench is for use in modeling redundant UUTs without instantiating them. The necessary handshaking can be accomplished in the test bench.

A dynamic test bench uses TEXTIO and file operations to simplify creation and to maximize run-time utility. Text files containing input stimulus are easy to work with using off the shelf software packages and can be modified without recompiling the VHDL test bench code. These files are parsed, and converted by the test bench to provide input stimulus during simulation. If changes are made to the text files, the simulation can be reset and run again without recompiling the test bench. As an example the output of some JTAG vector generation tools is in text format. Text output can be used to record status of run-time operations. Even greater flexibility is gained when the TEXTIO operations are used for run-time interfacing between the test bench and the user. The user inputs text file names or stimulus data directly into the test bench using the keyboard and receives prompting information and run time status via the screen. In this way, text files can be dynamically assigned, so that different test cases can be tried on the fly.

A segmented test bench is broken into individual test scenario sections with control flow mechanisms such that any combination of the these sections can be invoked as desired. Standard input from the keyboard or a control flow file are used to select which sections of the test bench will be exercised in any given simulation. Segmentation of the test bench allows the maximum flexibility of different types of test in the same compiled VHDL test bench without forcing the designer to wait for the completion of every test.

#### Section 4. Using the Test Bench in Gate Level Simulation

Taking advantage of the work done up front for the behavioral simulation is the key to moving quickly through gate level simulation. In fact, simulation might be too broad a word to describe the activity at the gate level- perhaps gate level verification is the right word. This was made possible by using state of the art mixed mode

simulation tools. A mixed mode simulation tool uses multiple simulation engines to interface the simulation of schematics, VHDL, and analog circuit descriptions within the same simulation. The methodology described was used by the authors to quickly create a correct design.

The authors were tasked with designing a high speed encryption device using bipolar differential logic. There were no synthesis target libraries available for the technology to be used and the schematic would be hand entered, so the utility of VHDL in modeling the circuit was initially in doubt. The authors overcame these difficulties by using VHDL as a test generation tool and for verification. As it turns out, the methodology is just as useful for synthesized designs. This methodology was made possible by using new mixed level simulation tools.

After a hierarchical block of gate level schematic is created, a work bench schematic is entered. This work bench includes the corresponding VHDL test bench as a component as well as the gate level UUT. The test bench is connected so that it drives the UUT inputs, and the UUT outputs can be connected for interactive monitoring by the test bench. An easy way to facilitate verification is to instantiate the VHDL behavioral model of the UUT and stimulate the two UUTs in parallel. The outputs of the behavioral and the gate level UUTs are compared by XOR circuits to alert the simulator and designer to any discrepancies between the two models.

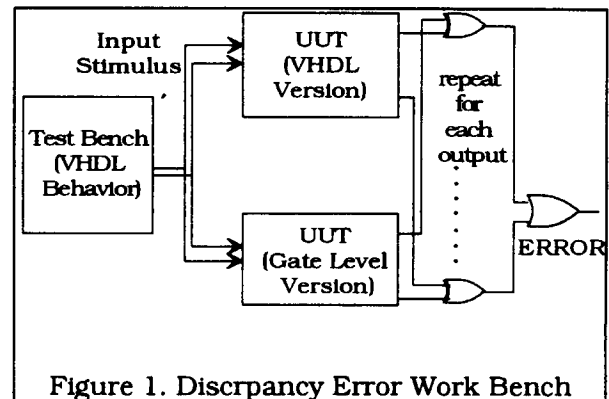
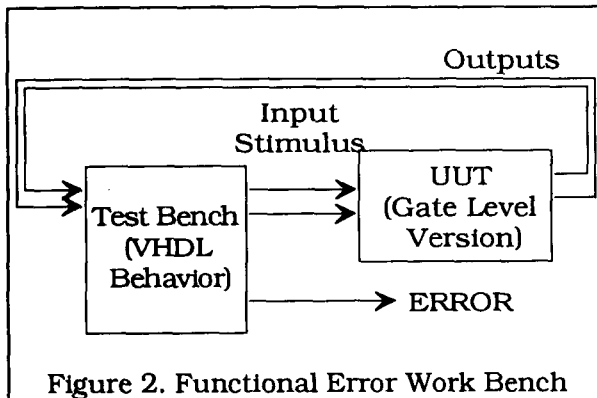


Figure 1. Discrepancy Error Work Bench

Alternatively, the gate level UUT outputs are monitored by the test bench and some sort of comparison is done internal to the test bench. This error signal can then alert the simulator and designer to any functional errors in the UUT.



Use of both the discrepancy and functional error generation techniques were used on various **MOTOROLA** GSTG programs. This combined configuration creates a powerful simulation run time debugging tool. Simulation break points can cause the simulation to halt on either a functional or discrepancy error. The debugging features of the simulation tool set can be directly invoked at the location of the problem to flush it out. There is no post processing of outputs compared against expected outputs to figure out where the errors are.

In any configuration, the enhanced debugging facility and the automated unambiguous verification of the gate design to the VHDL system requirement specification are the strengths of this methodology. It can rapidly advance designs through the gate-level simulation and verification tasks. Test vector stimulus translation is avoided, there is no post-processing of expected outputs, and the system is portable (inherent with VHDL being an industry standard).

### Section 5. Methodology Drawbacks

No honest treatment of a new methodology would be complete with out addressing the

drawbacks to its use. There are two main areas that need to be addressed. The methodology requires more time and effort in the requirements definition and preliminary design phases. The methodology requires the availability of a mixed mode simulation environment and a large amount of simulation resources.

During the requirements definition, a VHDL behavioral model must be developed. Additionally the test bench must be created. The completeness of this test bench is critical to verifying that the model meets all of the requirements of the design. This is no trivial task, and requires time and experienced personnel. For many design flows the VHDL modeling can serve as the preliminary design phase. This worked well for the authors. The sign-off of the preliminary design was on the VHDL model, instead of traditional block diagrams.

To use the VHDL test bench and behavioral model in the gate level verification simulation, one needs a mixed mode simulation environment. Instantiating a VHDL behavioral model into the gate level simulation may as much as double the magnitude of the overall circuit complexity. Furthermore, adding a complex VHDL test bench further increases the simulation complexity. Increased complexity has a direct impact on the amount of processing resources that need to be applied to the simulation effort. The impact may be felt in increased time, increased hardware horse power, or better software.

### Section 6. JTAG TAP Controller Example

A detailed design example for the derived methodology follows. An IEEE Std 1149.1-1990 JTAG TAP controller was selected for this example due to its wide appeal in the development of components in modern designs. This first step in the methodology was to create a VHDL behavioral model of the circuit. This code is a simple description of the state machine as detailed in IEEE 1149.1-1990 as Figure 5-1.

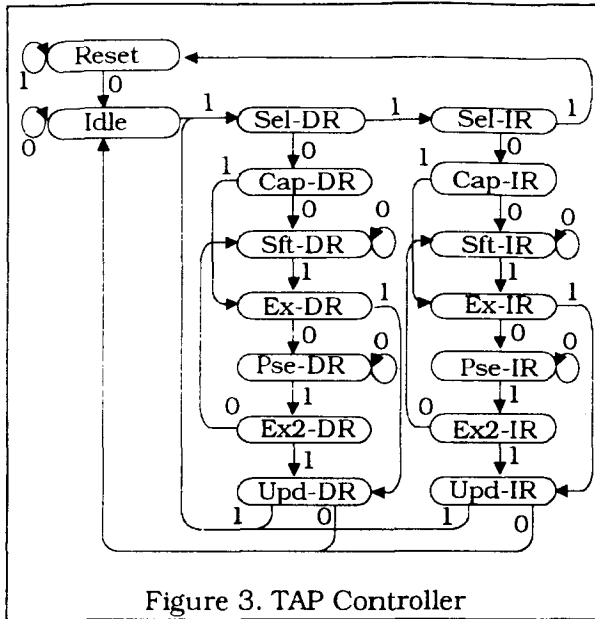


Figure 3. TAP Controller

The VHDL model consists of a case statement to create the state machine. Additionally there are some combinational logic circuits to create the outputs and the required clocks. The VHDL model is not listed in this paper.

The next step in the methodology was to create a test bench to stimulate the VHDL behavioral model. This code contains two processes. The first sets up a test clock for the simulation, the second is a flow control process that determines the tests to be run. The test bench uses the interactive, dynamic, and segmented techniques discussed above.

The flow control process is segmented by a switch statement so that different tests can be executed as desired. This switch statement is nested into a loop that surrounds the entire process. Each time through the process the user is prompted via the standard output as to the test that is desired. Standard input from the keyboard is then used to select which portion of the test bench will be exercised. The VHDL code for this function is structured as follows.

```

WHILE (done = 0) LOOP
-- choose flow
  printf("Enter Number of selection");
  printf(" 1. Parse TI ASSET File");
  printf(" 2. IEEE 1149 figure 5-7");
  printf(" 3. start-up tests");
  printf(" 4. Quit Test bench");
  READLINE(INPUT, in_line);
  READ(in_line, switch);

CASE switch IS
  WHEN '1' =>
    -- Parse TI ASSET spooler file
  WHEN '2' =>
    -- IEEE 1149 figure 5-7
  WHEN '3' =>
    -- start-up tests
  WHEN others =>
    -- Default quit
  END CASE;
END LOOP;

```

The "Parse a TI ASSET file" option will prompt the user for a Texas Instruments Advanced Support System for Emulation and Test generated spooler file that it will parse and inject the TCK, TRST\_L, and TMS vectors into the unit under test. An example ASSET spooler file is as follows:

```

# TTT
# MDD
# SIO
# Note: Initial spooler state is Test-
# Logic_Reset
#
#IR Scan
0 DDT
1 UDT
2 UDT
3 DDT
4 DDT
# .u1 (BCT8244)
5 DDH

```

The "IEEE 1149 Figure 5-7" option creates the TCK and TMS vectors to recreate the test as show in the IEEE Std Figure 5-7.

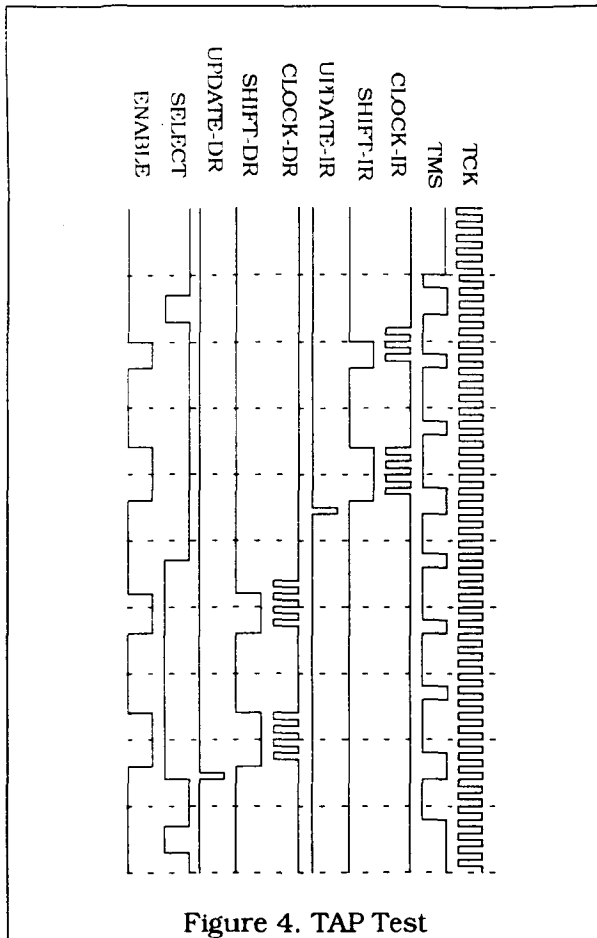


Figure 4. TAP Test

The "Start-up tests" option performs a reset and scans both the instruction register and the data register. The "Quit test bench" option exits the flow loop and ends the simulation.

Dynamic test bench techniques are used through-out the different test options. A procedure "printf" is created that takes any string as an input and uses access type operations to dynamically create storage for the array of characters and then send it to the standard output. This procedure is then called from anywhere that needs to send messages to the user.

```

-----
-- PROCEDURE Name: printf(text)
-- Description: Causes test bench to
-- output the text to the standard I/O
--
-- Author: Robert Yockey
-- Date: 4/14/93
-----

```

```

PROCEDURE printf (CONSTANT s: IN
string) IS
VARIABLE out_line: LINE;

BEGIN
    out_line := new string(1 to
s'length);
    out_line.ALL := s;
    WRITELINE(OUTPUT, out_line);
END printf;

```

A function "getfile" is created that prompts the user to enter a file name that is returned to the caller as a string of characters. This allows the user to specify file names at run time verses at compile time.

```

-----
-- FUNCTION Name: getfile() => string
-- Description: Prompts the user for a
-- file name and returns that file name
-- as a string
--
-- Author: Robert Yockey
-- Date: 4/14/93
-----

```

```

FUNCTION getfile RETURN STRING IS
VARIABLE file_name: LINE;

BEGIN
    printf("Enter Asset File Name: ");
    READLINE(INPUT, file_name);
    RETURN file_name.ALL;
END getfile;

```

A procedure "asset" was created to take a file name and parse through the file using TEXTIO commands to extract the test vectors from a ASCII file created by a independent test tool.

```

-----
-- PROCEDURE Name: asset(file_name)
-- Description: Causes testbench to
-- read a ASSET generated spooler file
-- and translate the stimulus to the
-- UUT. Assumes that the TAP begins
-- in Test-Logic-Reset
--
-- Author: Robert Yockey
-- Date: 4/14/93
-----

```

```

PROCEDURE asset (CONSTANT s: IN
  string) IS
  VARIABLE char: CHARACTER;
  VARIABLE in_line: LINE;
  FILE asset_file: TEXT IS IN s;

  BEGIN
    printf("Begin Asset Trans");

    READLINE(asset_file, in_line);
    read_file: WHILE (ENDFILE(asset_file)
      = FALSE) LOOP
      READ(in_line, char);
      IF (char /= '#') THEN
        read_tms: WHILE ((char /= 'D')
          AND (char /= 'U')) LOOP
          READ(in_line, char);
        END LOOP read_tms;
        IF (char = 'D') THEN TMS <= '0';
        ELSE TMS <= '1';
        END IF;
        wait_clk(1);
      END IF;
      READLINE(asset_file, in_line);
    END LOOP read_file;

    printf("End Asset Trans");
  END asset;

```

Next a work bench was created to test the VHDL behavioral model using the test bench. This created the interactive environment. The outputs of the test bench were used to stimulate the inputs of the unit under test, and the loop was closed by the outputs from the unit under test being routed to the inputs of the test bench.

The behavioral design was then verified to be correct using Synopsys Design Analyzer and Debugger tools on a Sun SPARC station 2. This completed the requirements definition part of the design.

The next phase of the design methodology was to create the actual circuit. Schematic Capture using the Mentor Design Architect environment on an HP 735 UNIX network was used to create the circuit. Mentor "Genlib" components were used to instantiate the required elements. The TAP controller circuit was directly lifted from the IEEE standard Figures 5-5 and 5-6. Both of which are show as follows:

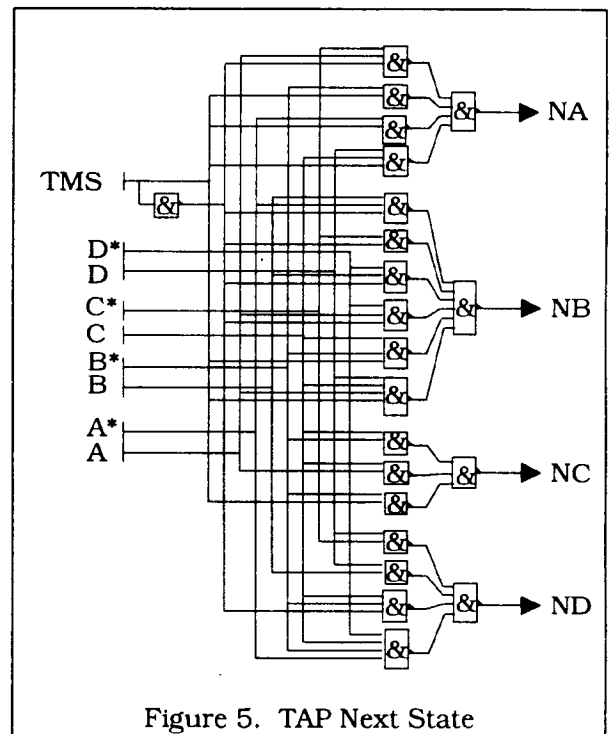


Figure 5. TAP Next State

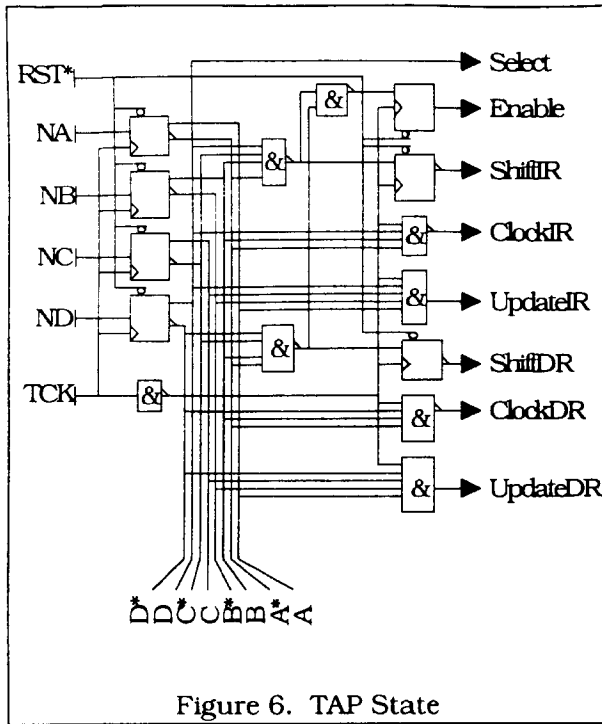


Figure 6. TAP State

Note that the example design does not implement the RESET signal and that the SHIFTIR and SHIFDR flops are made presetable not resettable on active TRST\_L.

The VHDL behavioral model of the TAP controller and the test bench that were developed in the requirements section of the methodology were imported into the Mentor Design Architect environment. Here the VHDL was recompiled by the Mentor tools. This machine independent characteristic of VHDL was very essential to our design environment, and can not be under emphasized.

All of the elements required are now in place to create the verification platform that is the focus of this paper. The Mentor Design Architect environment was used to instantiate the test bench and both the behavioral and actual model of the TAP controller onto a single schematic page. The outputs of the test bench were run to both of the models. The outputs of the two models were then sent to comparators (Mentor Genlib exclusive or gates) to create

error signals for each of the outputs. These were all combined (Mentor Genlib or gate) to create a system error signal. This error signal was then relocked by the test clock. This was so that timing differences between the two models would not cause glitching on the final error signal. Because the system is synchronous, one only cares that the outputs are stable and correct at the active edge of the clock. The outputs of the actual circuit were used to wrap back to the inputs of the test bench.

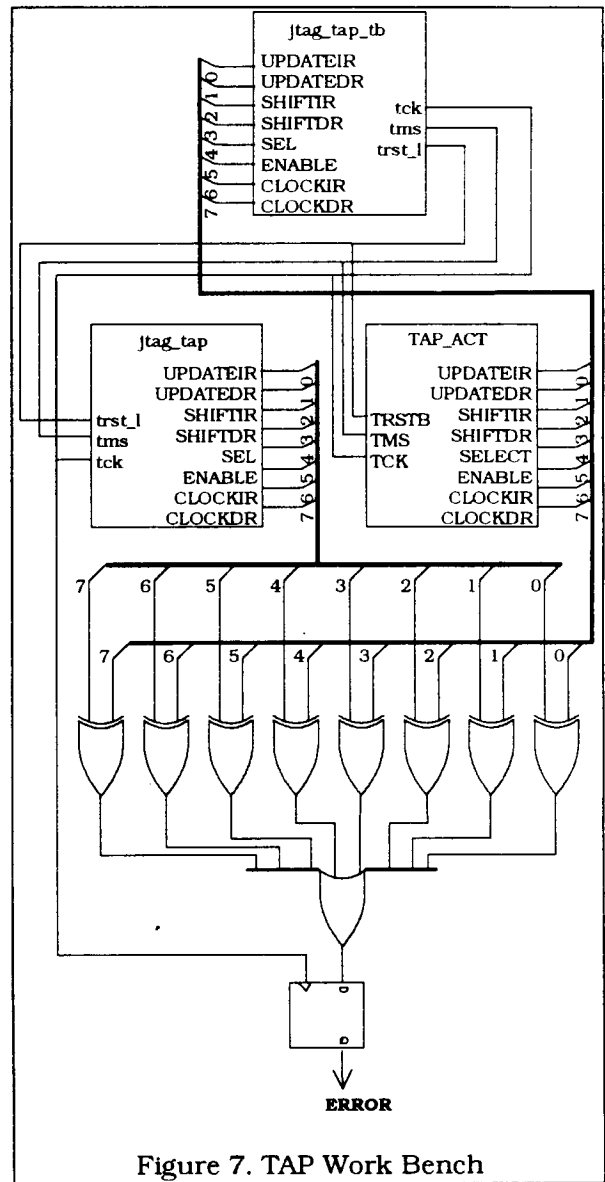
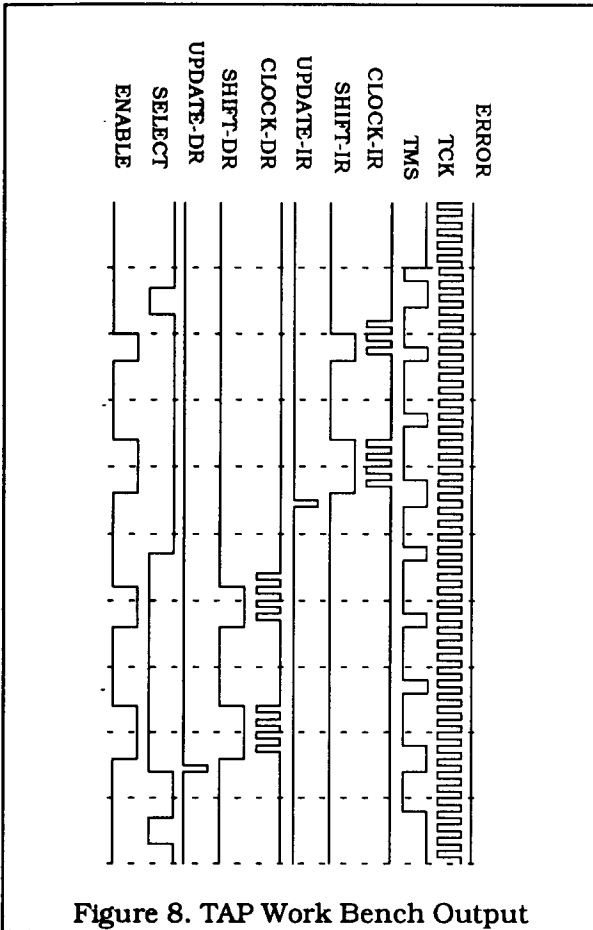


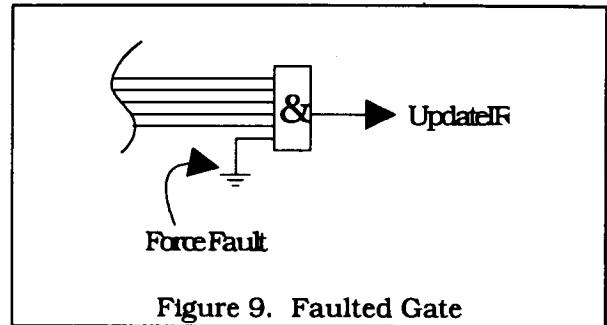
Figure 7. TAP Work Bench

Simulation and verification of the actual circuit was then performed. The Mentor QuickSimII tool was used to perform these tasks. Only the "IEEE 1149 Figure 5-7" option of the test was run to produce the output shown:

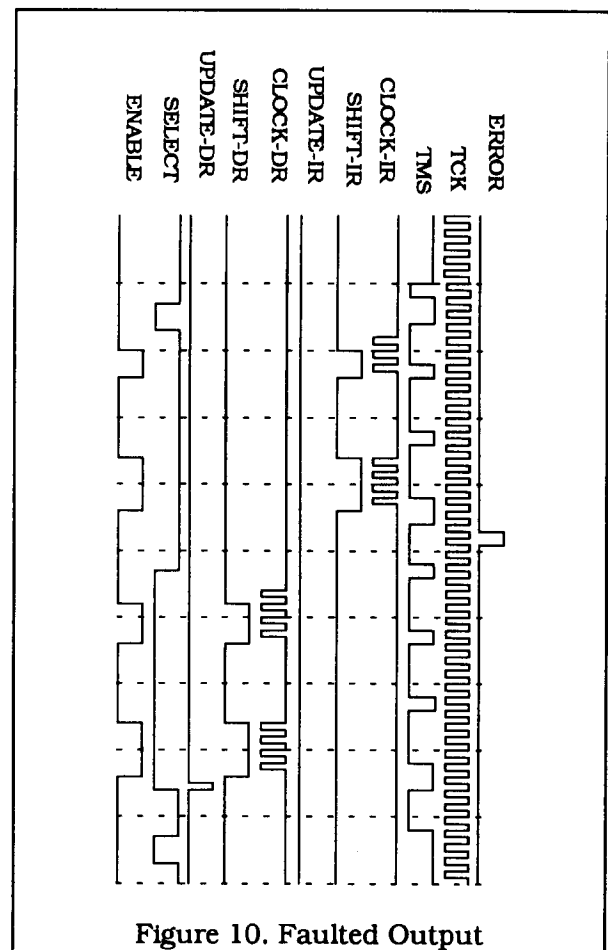


As expected with any simple example in a paper, the circuit works, which is not very interesting to real world debuggers.

During debugging of the actual circuit, the final system error signal was used as a simulation breakpoint to halt the simulation at points that the outputs of the two models differed. To show this point, an error was added to the actual TAP controller circuit. An extra input was added to the AND gate that creates the UPDATEIR signal that disables the gate. This errored circuit is shown:



Now when the "IEEE 1149 Figure 5-7" option of the test was run, it produced the following output:



Note now that the error signal shows that it had found an error. Stopping the simulation at a breakpoint on the error signal was found to be a very valuable tool for not only the actual debugging of this circuit, but also for work that was done on other programs. With the simulation

stopped at the point of the error, the examination powers of the Mentor tool set could be quickly brought to bear on correcting the problem.

## **Section 7. Summary**

Behavioral VHDL provides a modeling and documentation tool for system designers. Through use of the simulation techniques discussed, verification of a gate level design is greatly facilitated. Interactive techniques including dynamic file allocation were developed to further enhance the utility of the VHDL behavioral test bench.

A detailed example of the design and verification of a JTAG TAP Controller was discussed. The methodology was developed and used on **MOTOROLA** GSTG programs. The example included tools used and code samples.

By using the techniques described in the paper, designers avoid potential mistakes or lost time in verification. A communications link between systems designers and ASIC designers is established in the form of the behavioral circuit specification. Also, a tool for the system designer and design team to use in developing other parts of the system is created.

## **References**

IEEE Std 1149.1-1990 Standard Test Access Port and Boundary-Scan Architecture

Synopsys, Inc., "VSS™ Reference," 1098 Alta Ave., Mountain View, CA, 94043. (415) 962-5000.

Mentor Graphics, Inc., "Digital Systems Reference Manual" and "Design Architect™ Reference Manual" Version 8-1, 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070

Texas Instruments ASSET Scan-Based Diagnostics, User's Guild Ver 1.1