

Test Insertion Without Being a Test Expert

Mark T. Pronobis, Robert Hillman, and Christopher Flynn

Rome Laboratory
Design and Diagnostics Branch
525 Brooks Rd.
Griffiss AFB NY 13441-4505

Abstract

The use of automatic test insertion software and WAVES (Waveform And Vector Exchange Specification) can greatly reduce the need of VHDL designers to become test experts while still implementing efficient test solutions. This paper will describe the use of a test synthesis toolset to implement the IEEE 1149.1 boundary scan and internal scan into a in-house CMOS chip design that is being designed with VHDL. The use of WAVES and self-checking testbenches to test scan operations will also be described.

1.0 Introduction

A goal of this effort was to neither become an expert in Design for Test (DFT) nor an expert in microcircuit test, but still implement an efficient testability approach and a complete test solution (simulation to tester). The use of automatic test synthesis software from LV Software Inc. called ASICTEST, the use of WAVES, and the in-house development of WAVES tools helped to achieve this goal.

The ASICTEST toolset is an easy to use test synthesis toolset that allows a designer to describe and insert test elements like boundary scan, BIST (Built-In Self Test) for RAMS, and internal scan into an IC design at a high level. Test requirements are described in ASCII files which ASICTEST uses to produce VHDL or Verilog synthesizable RTL descriptions. These

test descriptions can then be synthesized with the rest of the design.

The use of WAVES and the in-house WAVES development tools reduced both design and test development time. Design time was reduced by adhering to the WAVES methodology of producing self-checking testbenches. Self-checking testbenches compare expected responses with actual responses making error detection easier and less error prone. The WAVES development tools generated the VHDL testbench and WAVES packages automatically from the top level entity declaration. The vectors created for simulation were also microcircuit tester compatible vectors thus reducing test development time.

2.0 Multiple Technology Processor (MTP)

A design of modest complexity was used for this effort. This particular design is being used to address several technical research areas, including Top-Down VHDL design, Top-Down WAVES methodology development, usage of WAVES in electrical testing, testability insertion and formal hardware verification.

The MTP design is a single-chip, 32-bit integer arithmetic processor (Figure 1). Resources include a 32-bit arithmetic logic unit, a thirty-two word register file and an internal clock generator/controller. The arithmetic logic unit is capable of executing the MTP's 26 instructions which include arithmetic (two's complement

signed), Boolean (bitwise), and conditional operations [1]. The data path is a quad-bus structure providing single cycle register data processing. The control, load, execute, and store states are implemented as single processor instruction cycles. The MTP is a static design and will be packaged in an 84 pin, Pin Grid Array (PGA).

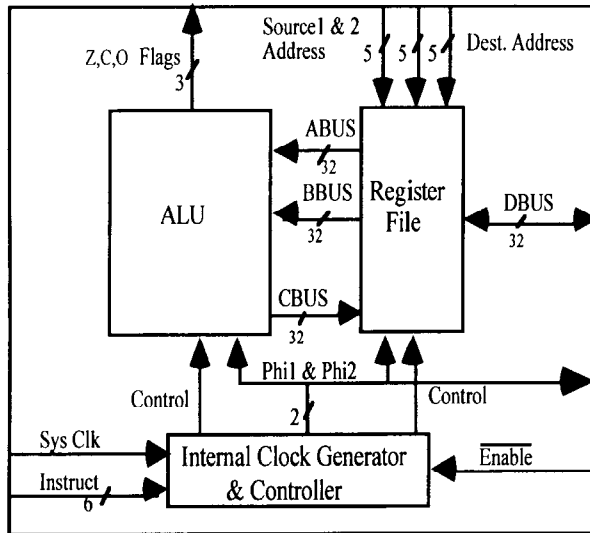


Figure 1. MTP Block Diagram

The MTP's register file, as mentioned above, offers thirty-two 32-bit general purpose registers. Registers R0 - R30 are all capable of being loaded with data from both the external interface Dbus and the internal ALU result driven Cbus. There is a "zero-register", R31, which is hard-wired to a default value of zero. The contents of this register can not be modified, and therefore always contains the value of zero. All 32 registers can be utilized for ALU operations, (i.e. read from, or written to, by ALU instructions). The internal clock generator/controller supplies the internal two-phase non overlapping clocks phi1 and phi2, as well as the required control signals for the ALU and register file. There are three internal 32-bit data busses, the A, B and C bus, and one 32-bit I/O bus, the D bus. There are three processor status flags, all active high. Further details of the MTP architecture, VHDL Modeling

methodology and Top-Down WAVES methodology can be found in references [2,3].

2.1 Testability Architecture

The testability architecture for the MTP includes Built-In Test (BIT), boundary scan and full internal scan to allow for the easy manufacturing verification of the fabricated chips. The boundary scan design for the MTP implements the IEEE-1149.1 boundary scan specification. The specification describes the design of a 16-state state machine, called the Test Access Port (TAP), that controls the operation of test resources that are on-chip. The TAP controls the internal scan path, the on-board Random Test Generator (RTG) and Signature Analyzer (SA).

The RTG provides random test vectors to be used for manufacturing level test of the ALU. The RTG will be implemented with a Linear Feedback Shift Register (LFSR) comprised of scannable flip-flops. The SA provides for the compression of the scan vectors to a 24-bit unambiguous signature implemented with a LFSR. The register file has externally accessible address and data pins and will be algorithmically tested from the external pins. A WAVES test set was developed to simulate the testability architecture in VHDL. Fault grading will be performed using gate level Hierarchical Integrated Test Simulator (HITS) [4] models reporting to the Mil-Std-883, Method 5012, "Fault Coverage Measurement for Digital Microcircuits" [5].

3.0 Test Insertion Automation

The JTAGSYN tool from the ASICTEST toolset was used to automate the insertion of the IEEE 1149.1 boundary scan into the design. Two separate text files are needed to describe the chip I/O and I/O pads as input to JTAGSYN. A simple text format is used to describe the chip I/O, internal scan chains, and the Device Identification Register. Figure 2 shows a portion of the input format for describing the chip I/O. The TRST pin describes an input pin that: (1) uses the padType X12IPD (input pad), (2) connects to the core input tcTRST, (3) is a

JTAG pin of type TRST, (4) is 1 bit wide, and (5) is connected to package pin 1. All I/O pins must be described in a format similar to this. Pin microA_in[1] describes a 4-bit input bus, pin microA_out[0] describes a 4-bit output bus and

```

options (MTP_jtag) {
  7/21/95/specify device ID
  revisionCode:      4'b0001;
  manufacturersIdCode: 11'h401;
  deviceIdCode:     16'h1234;
  technology:       LVS;
}

chip (MTP_jtag) {
  pin (TRST) {
    padType: X12IPD;
    toCore: tcTRST;
    jtagPinType: TRST;
    busWidth: 1;
    packagePin: 1;
  }
  .
  .
  pin (microA_in[0]) {
    padType: X12IPD;
    toCore: tcMicroA_in[0];
    busWidth: 4;
    packagePin: 7;
  }
  .
  .
  pin (microA_out[0]) {
    padType: OPAD12;
    fromCore: fcMicroA_out[0];
    busWidth: 4;
    packagePin: 22;
    enable: LogicHigh;
  }
  .
  .
  pin (CLK) {
    padType: X12IPD;
    toCore: tcCLK;
    busWidth: 1;
    isClock: YES;
    packagePin: 38;
  }
}

```

Figure 2. JTAGSYN I/O description

CLK describes the system clock. The isClock :YES attribute on the clock pin flags the software to place a sample only cell in the boundary scan for this pin. The sample only Boundary scan cell can also be used for any speed critical input pins.

An ASCII file describing the I/O pads, X12IPD and OPAD12, to be used in the design is also easy and straight forward to develop (Figure 3).

```

Pad definitions
padLibrary (cmosn)
pad(X12IPD) {
  direction: INPUT;
  port (Pad) {
    direction: input;
    connection: function(pinName);
  }
  port (out3) {
    direction: output;
    connection: function(fromPad);
  }
}

pad(OPAD12) {
  direction: OUTPUT;
  port (Pad) {
    direction: output;
    connection: function(pinName);
  }
  port (IN1) {
    direction: input;
    connection: function(toPad);
  }
}

```

Figure 3. JTAGSYN I/O Pad descriptions

JTAGSYN then uses these two descriptions to generate RTL VHDL code of the tap controller, boundary scan, instruction register, and the device identification register. The TAP also provides control and input and output to the internal scan. The RTL VHDL code of the test elements are easily synthesized and integrated to the rest design.

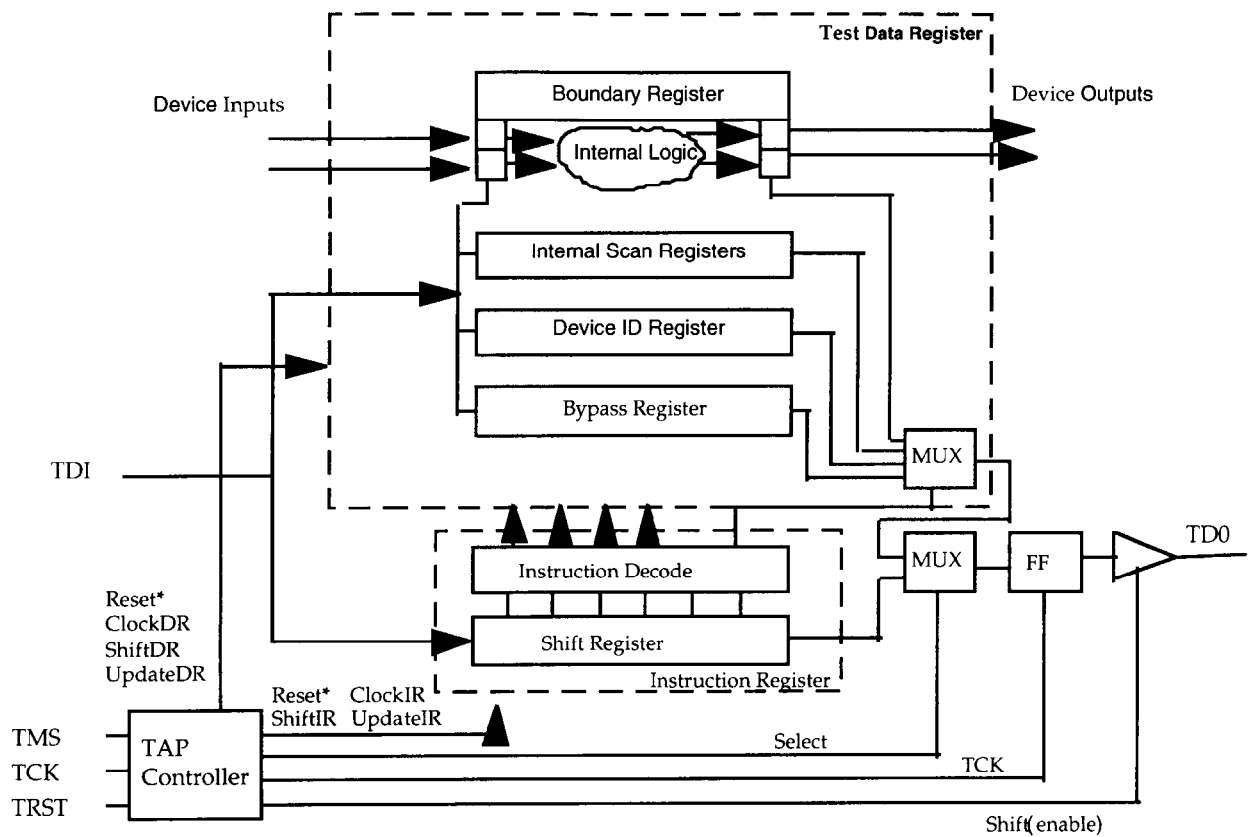


Figure 4. Boundary Scan Implementation

Figure 4 shows the boundary scan implementation that was automatically generated. All elements except for the internal logic and some instruction decoding have RTL VHDL descriptions that were automatically developed.

4.0 Waves Background

WAVES is IEEE Standard 1029.1-1991 for the representation of digital stimulus and response data for both the design and test communities. The WAVES standard format was developed to support users in the exchange of waveform information between different simulator and tester environments.

Because WAVES is an exchange specification, all facets of the stimulus and response data must be captured. Nothing must be left to the reader's imagination, for everyone has slightly different ideas about what constitutes expected data. What is "obviously right" to one

engineer is likely to be "obviously wrong" to another. When information is exchanged between environments, assumptions are dangerous and often incorrect. Therefore, WAVES data stands alone and does not require anything in common between sender and receiver, other than an adherence to the WAVES specification. WAVES is a subset of IEEE Standard 1076-1993 VHDL. Anyone with an understanding of VHDL will have a relatively easy time understanding the syntax and semantics of WAVES.

4.1 Waves Usage

The acceptance of a standard as a common practice is slow and requires an iterative process involving user demand, support tooling and user acceptability. Tool vendors will only invest in a tool development when it is apparent there is a profitable market available. Users on the other hand require both tool support as well as understanding of

the standard and the benefits of using that standard. The main ingredient to the adoption as a common practice is the availability of information for the use and application of the standard. The focus of this section is to discuss the work performed to support the use of WAVES with VHDL model simulation and verification.

A support library of pre-defined WAVES packages were developed to enhance the integration of WAVES and VHDL. This library provides a pre-defined environment for common applications of WAVES and VHDL. The hope is that these packages will be eventually integrated into the WAVES Language Reference Manual (LRM) [6].

In addition, a prototype automatic test bench generation tool has been developed based on these library elements. The purpose of the tool is to provide a methodology that minimizes the work involved by an individual by exploiting the use of a common user library. This methodology does not restrict the user, but rather provides an organized scheme to eliminate repetitive work and reduce the learning cycle for WAVES. The tool also automatically generates a testbench. The generated testbench not only applies WAVES stimulus to your model but verifies the validity of the expected responses.

4.2 Serial Vector Format

In order to support the development of the WAVES serial vectors to test and verify the boundary scan and internal scan an in-house tool was developed. This tool translates parallel vectors to serial vectors for the scan sections. The serial vectors of the external file are designated by a "S" while a parallel vector is designated by a "P". This was a crude attempt to develop a serial vector format and tool that supports IEE1149.1 operations and translates it into WAVES. As additional features were proposed, the in-house tool input format was converging to that of the Serial Vector Format (SVF) proposed by Texas Instruments [7]. The original serial scan format tool development was dropped and a tool that provides for the translation of high level 1149.1 bus operations based on the SVF into WAVES was initiated.

It is possible to use WAVES level 2 to translate the SVF format directly but then the translation of the vectors is done in-line with the execution of the simulation. It would also be fairly complicated to write the VHDL code to perform the translation. Therefore, Lex and Yacc [9] are being used for the translation of SVF to WAVES. The WAVES dataset then only has to read the translated external file and apply the serial vectors.

5.0 Rome Laboratory WAVES-VHDL Test Bench Tool

As mentioned above the purpose of the test bench tool is to aid a user in developing their WAVES data set and generation of a test bench for applying the stimulus and response to the models. This test bench tool was developed specifically for the application of WAVES and VHDL models conforming to the 1164 Standard Logic value system. Figure 5 is a simplified illustration of the test configuration that is generated. Stimulus waveforms that are produced by the WAVES data set are applied to the VHDL model, the model produces its' output response based on the stimulus presented. The test bench contains a set of monitoring processes that monitor the expected and actual response values and ensures that they also conform to the timing specified in the WAVES data set.

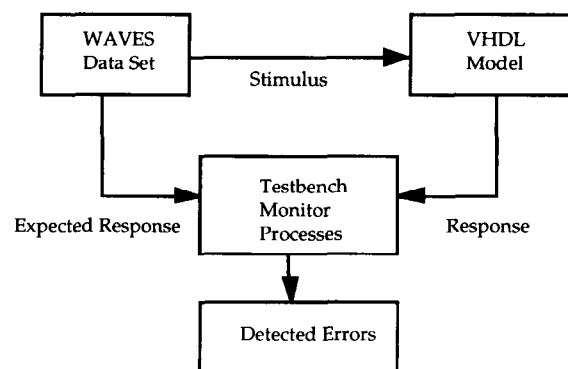


Figure 5. WAVES-VHDL Simplified Test Bench Configuration.

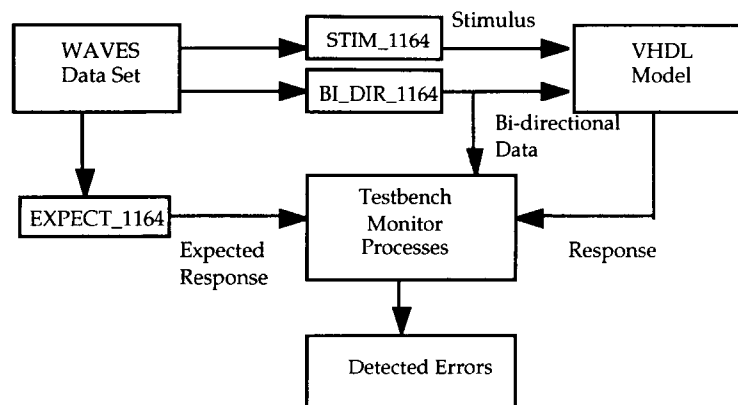


Figure 6. WAVES-VHDL Bi-directional Test Bench Configuration

5.1 WAVES Library

There are three different sets of library functions necessary to support the WAVES-VHDL test bench configuration shown above. The library functions address IEEE 1164 logic values, waveform shapes, and test bench utilities.

5.1.1 WAVES 1164 Logic Values

First, a WAVES logic value system was created. These WAVES packages define a logic value system based on the IEEE Standard 1164 - 1993 Multi-Valued Logic (MVL). They provide a library of reusable elements for WAVES users who are developing VHDL models that are compliant with the IEEE standard logic package. WAVES logic values define the events that occur on the waveform signals generated by the WAVES data set. This logic value system is almost identical to the Std 1164 logic values, however, they also account for the waveform signal direction which is required to generate a self monitoring test bench.

5.1.2 Waveform Shapes

A second set of library functions were created that establish a set of basic waveform shapes (formats) that can be used to construct complete, complex waveform descriptions for all the

input and output signals for a given model or unit under test (UUT). A WAVES waveform is comprised of three basic elements: pattern data (the truth table data), format (the waveshape), and timing (the edge transition points). These three elements combine to create the waveforms for data input to the model (the drive data) and the data expected on the output of the model (the expect or compare data). A library of functions that generate common shapes were constructed to simplify the waveform generation process and provide a link to ATE utilization. The library functions are used to specify the format and timing values associated with the supplied pattern data.

5.1.3 Test Bench Utilities

The third set of library functions were developed to provide for the integration and tool support in using WAVES in the VHDL simulation environment. This library of functions provides the means to connect the WAVES port list signals to the model and evaluate the model response for compatibility to the WAVES expected response. The library when utilized in conjunction with the test bench tool eliminates the hand development of the test bench. The functions generated have been developed to support a simple test bench with uni-directional pins as well as bi-directional pins. Figure 6 shows the testbench generated for a design with bi-directional pins.

5.2 Test Bench Generation Tool

In trying to make WAVES more helpful to VHDL users, the initial complexity of WAVES and lack of tool support were identified as weak points in WAVES utilization. Library elements were developed to reduce the learning curve and provide a set of library functions that will meet 90% of all users needs. Second a tool was developed that utilizes WAVES as a standard approach for model verification and eliminates the need for user generated code for supporting intelligent test verification.

The test bench generation tool has several features that were developed to aid the user when using WAVES and VHDL together. First, the tool generates WAVES packages that are utilized in defining the model specific WAVES data set. The UUT_pins package, waveform generator procedure template, and header file template are the WAVES model specific packages generated, they will be discussed latter. Second, the tool will perform a syntax check on the external pattern file to be utilized with the model. It is required that the user create or have this file generated in conformance with the WAVES level 1 syntax [6]. Finally, the tool will generate the test bench that wires the WAVES waveform procedure previously generated and the model together for design verification. The main input to the tool is the model entity declaration. The entity declaration is parsed by the tool and used to define all appropriate signals and data structures for creation of the WAVES data set and the VHDL test bench code.

The WAVES UUT_pins package developed by the tool is automatically generated and requires no user modifications. The only important fact a user must know is that this package defines the order or the pin assignments associated with the columns in the external pattern file. The waveform generator procedure template provides a basic structure based on the model to define pin groupings and signal association for waveform shapes. The tool makes a first cut at establishing all required pin groupings and provides a template section for the user to supply the signal format and timing values associated with the external pattern data. The header file template is

generated to provide configuration information. The WAVES source files and order of analysis are automatically captured and all the user is required to do is add any site or model specific textual information deemed relevant.

5.5 Beta Tool Release

The tool is currently in Beta testing. The tool is being used at Rome Laboratory to support in-house model verification. It has been successfully used by four non-WAVES literate users and has met all development goals. In addition, the tool has been released to Martin Marietta Labs, Defense Electronic Supply Center, and MTL Inc. The tool has been successfully utilized at all three facilities with extremely positive response given. The tool will be provide to the public domain using the WAVES web reflector at VHDL.ORG as soon as all documentation and a set of user examples are completed.

6.0. Conclusions

The use of automated test insertion software greatly reduces the need of circuit designers from becoming an expert in testability algorithms, test element implementations, and testability approaches. The structured approach to test insertion that this software provides greatly reduces design time while implementing a verified, efficient test solution.

WAVES should not be looked at as costing something extra in the design cycle The opposite is quite true. First, WAVES is nothing more then VHDL. Second, the methodology for using WAVES requires that the responses be supplied with the stimulus. The benefit here is that the designer is required to think up front about the responses and not after the simulation is complete by looking over stacks of simulation reports. The latter approach is very tedious and error prone. With WAVES you know when the expected and actual response do not match. Third, with WAVES there is no need to translate design vectors to test vectors since the wave shapes used during simulation are the same shapes that the tester can accept. The lack of tool automation has hampered the

acceptance of WAVES in the design and test of VHDL designs. The development of site libraries, standard packages, and the tools presented in this paper are a good step to solving this problem.

There are several additional tasks that are planned for the future. The first is to complete the SVF compiler tool. The second task is to demonstrate how the WAVES test bench tool can be used to support algorithmic type test sets rather than using a pattern file. We are also in the process of performing an evaluation of the performance characteristics when utilizing different testbench approaches with WAVES.

References

- [1]. Flynn, C.J., Hall, F.G., Hanna, J.P. and Hillman, R.G., "Multiple Technology Processor (MTP) Specification, Draft, 20 June 1995, Rome Laboratory Technical Report, to be published.
- [2]. Flynn, C.J., Hall, F.G., Hanna, J.P., Nassif, M.P., and Pronobis, M.T., "Top-Down Design Through Test using VHDL and WAVES," Proceedings 1995 International Conference on Electronic Hardware Description Languages (ICEHDL), January 1995, pp. 21-25.
- [3]. Flynn, C.J., Hall, F.G, Hanna, J.P., and Pronobis, M.T., "Using WAVES in a Top-Down Design Methodology," Proceedings VHDL International Users' Forum, Fall Conference, November 1994, pp.12.1-12.6.
- [4] L. Hosley and M. Modi, "HITS - The Navy's new DATPG system," Proceedings, AUTOTESTCON, 1983.
- [5] Warren H. Debany JR., Kevin A. Kwait, and Sami A. Al-Arian, "A Method for Consistent Fault Coverage Reporting," IEEE Design & Test of Computers, September 1993.
- [6]. "IEEE Standard for Waveform and Vector Exchange (WAVES)," Institute of Electrical and Electronics Engineers, IEEE Std 1029.1-1991.
- [7] "Serial Vector Format Specification," Texas Instruments Inc., Revision B, 1992.
- [8] John R. Levine, Tony Mason, and Doug Brown, "lex and yacc," Oreilly & Associates, Inc., 1992.