Using WAVES For VHDL Model Verification

James P. Hanna
Electronics Reliability Division
Rome Laboratory
Griffiss AFB, NY 13441-4505
hannaj@rl.af.mil

ABSTRACT

This paper outlines and provides an example approach to developing a standard WAVES test bench for VHDL model verification. Issues regarding the use of WAVES in a VHDL simulation environment will be discussed. This includes how and where to obtain the WAVES standard packages, setting up the WAVES_STD library, and compiling the WAVES standard packages with the VHDL model. Next, an example of a self-monitoring WAVES test bench is presented and discussed. Finally, simulation efficiency issues and some conclusions are discussed.

INTRODUCTION

WAVES, the Waveform And Vector Exchange Specification, is the IEEE standard exchange format for the capture and exchange of both simulation histories and test vector sets. WAVES is a subset of VHDL, and therefore can be compiled (by any VHDL compiler) and used as the source of stimulus and expected response for VHDL model verification.

The development of the WAVES standard was sponsored by two separate IEEE committees. The Automatic Test Program Generation (ATPG) working group of the IEEE Standards Coordinating Committee 20 (SCC20) was interested in developing a standard format for describing test vectors. The Design Automation Standards Committee (DASC) was interested in standardizing VHDL usage for test. Together these two committees sponsored a working group with the charter of finding a common solution to both sets of requirements.

The effort of the WAVES working group was supported by international volunteers with expertise in digital simulation and test from a wide variety of backgrounds. The experience base ranged from Computer Automated Design (CAD) tool developers and users, Automated Test System (ATS) developers, electronic device manufacturers, and System OEMs.

WAVES was adopted in Dec 1991 as IEEE standard 1029.1-1991. Until the development of the WAVES standard, there was no common format for the representation of stimulus and response information for digital microcircuits. Manufacturers of commercial computer aided design, simulation and automated test systems all used their own proprietary formats for representing the stimulus and response information and as such this information was not easily ported between the design community or between the test community or between the design and test communities.

The WAVES format for representing timing and vector information can be used with VHDL for simulation and design verification. This paper describes how the WAVES standard can be applied to the process of design verification for VHDL models. The approach taken in applying WAVES in this paper is to use the WAVES data set in conjunction with a VHDL test bench to provide a virtual test environment for the VHDL model.
A WAVES data set is the collection of all WAVES declarations necessary to fully describe the waveforms for a set of input and expected output signals for a VHDL model. A WAVES data set consists of the following WAVES declarations and ancillary files:

1) the header file,
2) the test pins type declaration,
3) the logic values type declaration,
4) the pin codes constant declaration,
5) the value dictionary function declaration
6) the frame set declaration,
7) the external pattern file,
8) the waveform generator procedure.

```
-- This is an example of a WAVES
-- header file for a simple 4-bit
-- synchronous up counter with
-- a synchronous clear.
--
TITLE           4-Bit counter example
DEVICE_ID       Example
DATE             02-06-1995
ORIGIN          Example
AUTHOR           Rome Laboratory ERDD
AUTHOR           James P. Hanna
--
-- Data set construction.
--
WAVES_FILENAME  test_pins      WORK
WAVES_FILENAME  logic_val      WORK
WAVES_FILENAME  pin_codes      WORK
use WORK.test_pins_pkg.all;
use WORK.logic_val_pkg.all;
use WORK.pin_codes_pkg.all;
WAVES_UNIT       WAVES_INTERFACE WORK
WAVES_FILENAME  value_dict     WORK
WAVES_FILENAME  frames         WORK
use WORK.test_pins_pkg.all;
use WORK.logic_val_pkg.all;
use WORK.pin_codes_pkg.all;
use WORK.WAVES_Interface.all;
WAVES_UNIT       WAVES_OBJECTS  WORK
WAVES_FILENAME  generator      WORK
--
EXTERNAL_FILENAME patterns TEST
WAVEFORM_GENERATOR PROCEDURE
WORK.WGP.Waveform
```

Figure 1. Header File.

A WAVES data set is most useful when partitioned and organized for maximum reuse. This can be accomplished by placing each required WAVES in its own package. This results in several files being generated for each WAVES data set. This section will discuss the required WAVES declarations and show how they are assembled into a complete WAVES data set.

The Header File

Each WAVES data set must have a header file. The header file provides a mechanism for documenting the contents of the data set, configuration management information, and the proper order of compilation for the components of the WAVES data set. The header file consists of two sections: configuration information, and contents and compilation order. An example of a WAVES header file is given in figure 1. For complete details on the syntax of the WAVES header file refer to the WAVES Language Reference Manual (LRM) [1].

Test Pins

The names of all of the pins on the interface of the VHDL model must be declared in the WAVES data set. This declaration is a simple enumerated type declaration in WAVES. The name of the type must be "Test_pins." To separate device dependent information from reusable information this declaration should be placed in its own file. An example of a test pins declaration is given in figure 2.

```
Package test_pins_pkg is

  type Test_pins is
    (CLK, CLK, O3, O2, O1, O0);

end test_pins_pkg;
```

Figure 2. Test Pins.
The order of the enumeration of the names of the test pins declaration indicates the ordering of the signals in the WAVES port list and the column ordering of the pin codes in the external pattern file. The WAVES port list and the external pattern file are discussed later.

Logic Values

The names that will be used to describe the logic values of events that occur on a given signal must be declared. This declaration is a simple enumerated type declaration in WAVES. The name of this type must be "Logic_value." This declaration, as we will see later, can be reused and therefore should be placed in its own. An example of a logic value declaration that models all of the IEEE 1164 simulation code values that are relevant to test is given in figure 3.

```
package logic_val_pkg is

type Logic_value is
  ( DONT_CARE, HIGH_IMPEADANCE, FORCE_0, FORCE_1, SENSE_0, SENSE_1 );

end logic_val_pkg;
```

Figure 3. Logic Value.

WAVES can distinguish between events that are driven into the VHDL model and events that are driven by the VHDL model. The way that this is accomplished using WAVES is to define logic values for both the case of driving and sensing '0' or '1' values. In this paper the WAVES data set is being used to describe the stimulus being applied to the model as well as the response predicted at the model outputs. This is why the logic value declaration in figure 3 enumerates two additional values for sensing the outputs of the model. The intent of the data set is that applying the stimulus to the model inputs will result in the expected response at the model outputs given that the model behaves correctly. Used in this manner, the WAVES data set can be thought of as a virtual tester.

WAVES models events on the waveform as the set (state, strength, direction, relevance). This differs from IEEE 1164 in that 1164 does not account for direction or relevance. The richness of the WAVES event model is due to the fact that the WAVES standard is intended to support multiple simulation and electronic test environments.

For VHDL simulation, this model need not be fully specified; only is required to be elaborated. However, as the detail of a given design expands additional information can and should be added to the WAVES data set. For instance, the determination of the bidirectionality of a given pin should be annotated in the WAVES data set.

Pin Codes

The character codes that will be used to represent the ones and zeros in the external pattern file must be defined in a constant declaration. This constant declaration must be named "Pin_codes." These character codes can be any printing character including any letter, digit, or special character. The order of the declaration of character codes is not significant in the case of this example because of the manner in which the frame sets are defined later.

```
package pin_codes_pkg is

constant PIN_CODES : String := "01z-";

end pin_codes_pkg;
```

Figure 4. Pin Codes.

This flexibility in defining the characters used in the external pattern file allows WAVES data sets to be quickly constructed.
from the output of any simulator or modern Automatic Test System (ATS).

This declaration should also be placed in its own file. An example of a pin codes declaration that uses all of the IEEE 1164 codes that are relevant to test is given in figure 4.

Value Dictionary

```
function Value_Dictionary( VALUE :
    Logic_value ) return
    event_value is
begin
    case VALUE is
    when DON'T_CARE =>
        return UNSPECIFIED;
    when HIGH_IMPEDANCE =>
        return state = MILIBAND
        and strength = CAPACITIVE;
    when FORCE_0 =>
        return state = LOW
        and strength = DRIVE
        and direction = STIMULUS;
    when FORCE_1 =>
        return state = HIGH
        and strength = DRIVE
        and direction = STIMULUS;
    when SENSE 0 =>
        return state = LOW
        and direction = RESPONSE
        and relevance = REQUIRED;
    when SENSE 1 =>
        return state = HIGH
        and direction = RESPONSE
        and relevance = REQUIRED;
    end case;
end Value_Dictionary;
```

Figure 5. Value Dictionary.

This function must be named "Value_Dictionary." The value dictionary declaration should be placed in its own file. An example of a value dictionary function declaration for supporting the IEEE 1164 standard is given in figure 5.

The details of the event value data structure and its use by the WAVES data set is hidden from the user. Therefore, it suffices to understand only that the value dictionary simply provides meaning to the names chosen for the logic value declaration. The value dictionary is used only for documentation in WAVES; it is not used for actual event generation.

Frame Sets

```
function Non_Return( T1 : Time )
return Frame_set is
    constant EDGE : Event_time :=
        Etube( T1, 250 μs );
    constant NO_EVENT : Frame :=
        Frame_Event;
begin
    return
    New_Frame_Set( '0',
        Frame_Event((FORCE_0, EDGE)) ) +
    New_Frame_Set( '1',
        Frame_Event((FORCE_1, EDGE)) ) +
    New_Frame_Set( 'Z',
        Frame_Event((HIGH_IMPEDANCE, EDGE)) ) +
    New_Frame_Set( '-', NO_EVENT );
end Non_Return;
```

Figure 6a. Non Return Frame Set.

Typically, a simulation pattern file represents the ones and zeros as single events occurring on the given signals in the given simulation cycle. This representation assumes that the stimulus events are driven at the beginning of the cycle and remain in effect until the end of the given cycle. Fixed strobe times, usually near the end of the simulation cycle, are assumed for comparing the actual model response to its expected response.

6.14
A typical modern ATS provides a set of waveform "shapes" or formats that are used to provide stimulus events to the Unit Under Test (UUT). These shapes may, and often do, allow the representation of multiple events in a given test cycle. These formats are typically non-return, return to zero, surround by complement, etc., and may include a format for generating a multiple phase clock. The codes in the tester pattern files, then, represent different sets of events depending upon which tester format is used by a given signal on the UUT.

WAVES provides a construct called the frame set which allows the pin codes in the external pattern file to be interpreted as contributing multiple events to the given waveform; for example, multiple cycle clocks.

Figure 6b. Pulse High Frame Set.

Figure 6c. Compare Frame Set.

A frame set is a collection of formats that are associated with a set of pins for the generation of a waveform. In the most simple case, a frame for a given pin will have one event that occurs at the beginning of a given simulation cycle. This is how a typical simulation pattern file is represented in WAVES. Figures 6a - 6c show three WAVES functions that generate a simple single event format, a single phase clock, and a compare window. These frame set functions are associated with pins in the waveform generator procedure, discussed later. Figure 7 provides a graphical representation of these frame definitions. These functions are used in the waveform generator procedure to define the appropriate interpretation of the pin codes in the external pattern file for the pins of the model of the counter. These frame declarations should also be placed in their own file.
External Pattern File

```
% Example external pattern file for %
% a simple synchronous 4-Bit up %
% counter with synchronous clear. %
% %
% C C 0000 %
% L L 3210 %
% K R %
% %
1 1 ---- : 200 ns;
1 0 0000 : 200 ns;
1 0 0001 : 200 ns;
1 0 0010 : 200 ns;
1 0 0011 : 200 ns;
1 0 0100 : 200 ns;
1 0 0101 : 200 ns;
1 1 ---- : 200 ns;
  
```

Figure 8. External Pattern File.

The external pattern file contains a truth table representation of the waveforms for all of the input signals and the expected response of all output signals. Each row of this file consists of one character code for each pin named in the test pins declaration. The order of the columns in the external file is identical to the order of the pin names in the test pins declaration. In addition to the character codes, each row may contain an optional delay time. The optional delay time, if present, indicates the amount of time that the waveform generator procedure is to delay before applying the next row of the external pattern file to the VHDL model.

The waveform generator procedure is discussed in the next paragraph. The external pattern file has additional options besides those mentioned here. For a complete description of the syntax of the external pattern file see the WAVES LRM [1]. An example of an external pattern file based on the pin codes declaration in figure 4 is given in figure 8.

Waveform Generator Procedure

The waveform generator procedure (WGP) is the procedure that brings all of the required WAVES declarations and the external pattern file together. This procedure uses the required WAVES declarations to interpret the patterns in the external file. An example of a waveform generator is given in figure 9.

The WGP reads each row of the external pattern file one at a time and applies the character codes as signals to its interface parameter. This interface parameter, known as the WAVES port list, is a special data structure that provides the signals that will be connected to the VHDL model in the WAVES test bench. The waveform generator procedure is used by the test bench as a concurrent procedure call. The WGP then, is a concurrent VHDL process that executes once for each row of the external pattern file.

A SITE LIBRARY

At first glance, it may seem that developing a WAVES data set is a complicated task. In reality, though, the task of generating a suitable set of tests is far more arduous than documenting the test set with WAVES. Indeed, having a standard format for documenting test sets for VHDL model verification allows the user to concentrate on the content of the tests rather than their capture and formatting.
procedure Waveform( signal WPL : inout WAVES_port_list ) is

-- File and slice variables.
--
file EXTERNAL : Text is in
  "patterns";
variable SLICE : File_slice :=
  New_File_Slice;
--
  Read File placement data.
--
constant T0 : Time := 0 ns;
constant T1 : Time := 50 ns;
constant T2 : Time := 100 ns;
constant T3 : Time := 150 ns;
--
  Frame set array and Timing data
--
constant PSA : Frame_set_array :=
  New_Frame_Set_Array(
    New_Frame_Set_Array(
      Non_Return( T0 ), CLR )
    ) +
  New_Frame_Set_Array(
    Compare( T2, T3 ) ,
    ( 01, 02, 01, 00 ));
variable TIMING : Time_data :=
  New_Time_Data( PSA );
begin
  loop
    Read_File_Slice( EXTERNAL,
                     SLICE );
    exit when Endfile( EXTERNAL );
    Apply( WPL, SLICE.Codes.all,
           Delay( SLICE.FS_TIME ),
           TIMING );
  end loop;
end Waveform;

Figure 9. WGP.

Much of the information that comprises a WAVES data set can be reused in its entirety. The pin codes declaration, the logic value declaration, the frame set declaration, and the value dictionary declaration need only be defined once. Once these WAVES declarations have been defined and placed in packages they should be compiled into a separate WAVES site library. Every WAVES data set that is developed subsequently only needs to reference these reusable WAVES packages that are stored in the WAVES site library. The only parts of the WAVES data set that will need to be developed by each user will be the test pins declaration, the external pattern file, the waveform generator procedure, and the header file. In most cases, the waveform generator procedure will remain unchanged from data set to data set (with the exception of assigning the frames that are to be used for specific pins).

The WAVES site library declarations should be agreed upon by the VHDL designers that will be generating WAVES test benches to verify their models. These packages should be developed by your site’s WAVES expert and take into consideration the modeling codes that your site will use, the set of logic values that will be used, and the frames that will be used at your site.

WAVES STANDARD LIBRARY

In order to compile your WAVES data set in your particular VHDL environment you must have the WAVES standard packages compiled into a library named “WAVES_STD.” The WAVES standard packages are a VHDL implementation of the WAVES Language Reference Manual (LRM). These packages contain code for all of the standard WAVES functions, procedures, and data types that specify the unique semantics of the WAVES language.

The WAVES standard packages can be obtained through anonymous ftp from vhdl.org. The packages are available in Unix tar format, Unix compressed tar format, or GNU zip tar format and are stored in the /vi/waves/ftp_files directory.

The tar release for these packages includes a README file that describes how to set up the WAVES standard library and compile the standard packages, as well as a RELEASE.NOTES file that discusses all known bugs in the implementation.
The packages that are included with the tar release are: WAVES_System, WAVES_Standard, WAVES_Interface, and WAVES_Objects. The WAVES_System and WAVES_Standard packages must be compiled into a library named "WAVES_STD." The WAVES_Interface and WAVES_Objects packages must be compiled in the appropriate order along with the user defined elements of your WAVES data set. The steps required to compile the WAVES data set and the contents of the local design library, are discussed in the next section.

LOCAL DESIGN LIBRARY

The contents of your local design library are very similar to those for a typical VHDL design project. The VHDL model resides here along with the test bench and any support packages. The main difference is that this library contains several packages that are required by the WAVES data set. These include the WAVES data set itself as well as the two WAVES standard packages: WAVES_Interface and WAVES_Objects.

These two WAVES packages contain the portions of the WAVES implementation that are dependent on your specific WAVES data set. The test pins, logic value, and pin code declarations must be made visible to these packages during compilation. This is because these packages declare data structures that are needed by the implementation. The size of these data structures can only be determined during compile time. For example, the WAVES port list for your particular WAVES data set will depend on the size and order of the test pins declaration. Also, the test pins declaration is dependent on the interface to the VHDL model that is being tested.

The order of compilation for a WAVES data set is given in the corresponding header file. This file indicates which files make up the entire waves data set as well as the order in which they are to be compiled. Consider the example header file given in figure 1.

In the data set construction portion of this file we are first instructed to compile the WAVES_FILENAMEs and the WAVES_UNITS in the given order. We first compile three WAVES_FILENAMEs: test_pins, logic_val, and pin_codes. Next we compile the WAVES_UNIT WAVES_Interface in the context of the context (use) clauses that precede it. Next, we compile the two WAVES_FILENAMEs: value_dict and frames. Then, we compile the WAVES_UNIT WAVES_Objects in the context of the context clauses that precede it. Finally, we compile the WAVES_FILENAME generator. All of the packages contained in these files are compiled into the current working library, as is indicated by the library name "WORK" that occurs at the end of each line.

The occurrence of context clauses in the header file indicates that all of the context clauses that directly precede a WAVES_FILENAME or WAVES_UNIT keyword, are to be prepended to the file given after the keyword. After the context clauses have been prepended, the file is then compiled. This assures that these packages are made visible to the WAVES_FILENAME or WAVES_UNIT so that the file will compile correctly.

The Test Bench

Once you have developed your VHDL model and your WAVES data set all that remains is to develop a simple test bench program to apply the stimulus from the WAVES data set to your model and compare the expected response to the actual response of the model. The test bench program consists of the following parts: local signal declarations, a concurrent procedure call to the waveform generator procedure, a process to translate the WAVES logic values into 1164 signals, the port mapping of the component to the appropriate signals, and a set of processes that monitor the actual results and the
expected response and report any discrepancies.

entity Testbench is end Testbench;
architecture WAVES of Testbench is
--
 Signal declarations.
--
signal WPL : WAVES_port_list;
signal O3_ACTUAL : Std_logic;
signal O3EXPECTED : Std_logic;
 end

Figure 10a. Test Bench Signal declarations.

Figure 10a through 10e shows a fragment of code for each of the segments mentioned above. The signal declaration section, figure 10a, must have one signal of type WAVES_port_list to receive the stimulus and expected response from the WAVES data set. In addition, one signal should be declared of type std_logic for each element of the WAVES port list (remember that the elements of the WAVES port list correspond to the test_pins declaration in number and order). The concurrent procedure call in figure 10b is straightforward.

begin
--
-- WAVES process.
--
WAVES : Waveform( WPL );

end

Figure 10b Concurrent Procedure Call.

The translation process in figure 10c demonstrates how the WAVES logic values are translated into 1164 signals to be applied to the model. This process is sensitive to the WPL signal generated by the concurrent procedure call. The translation uses a lookup table (To1164) that maps the WAVES logic values to the appropriate 1164 values.

XLATE: process( WPL )
begin
  CLK <= To1164(
    Logic_value'val( 
      WPL.WPL( 1 ).L_VALUE ));
  CLR <= To1164(
    Logic_value'val( 
      WPL.WPL( 2 ).L_VALUE ));
  O3EXPECTED <= To1164(
    Logic_value'val( 
      WPL.WPL( 3 ).L_VALUE ));
end process;

Figure 10c Translation Process.

The port mapping in figure 10d is standard VHDL code. Notice that we mapped the Ox_ACTUAL signals to the output ports of the model. Later we will compare these signals to the OxEXPECTED signals in the monitor processes.

U1 : Four_Bit_Counter
 port map(
  CLK => CLK,
  CLR => CLR,
  O3 => O3_ACTUAL,
  O2 => O2_ACTUAL,
  O1 => O1_ACTUAL,
  O0 => O0_ACTUAL );

Figure 10d Port Mapping.

The monitor processes in figure 10e are sensitive on both the actual response of the model and the expected response as given by the WAVES data set. Whenever these two signals differ an assertion violation occurs and an error message is reported. The compare function used by the monitor
processes returns the value true whenever
the WAVES data set indicates the
expected response of "don't care" ('-').

```
MONITOR_03 : process ( 03EXPECTED,
                        03ACTUAL )
begin
  assert ( Compare ( 03EXPECTED,
                     03ACTUAL ) );
  report "-> Error in output 03."
  severity WARNING;
end process;

Figure 10e  Monitor Process.
```

CONCLUSIONS

The WAVES standard, although not
primarily developed for use with VHDL
for model verification, is very useful in this
capacity. WAVES provides a standard
way of expressing the stimulus and
expected response for VHDL model
verification. Once a user learns how to
represent test sets using WAVES for one
modeling effort, it is simple to apply this
knowledge to subsequent modeling projects.
An additional benefit is the unambiguous
documentation of the test set in a manner
that can be readily transported to
electronic test.

There are some drawbacks associated with
using WAVES for VHDL model
verification though. One is that the
VHDL modeler must learn to represent test
sets using the WAVES language. This is a
minor issue. A more serious concern is that
of simulation efficiency. Simulation
efficiency suffers due to the use of waves for
two reasons: 1) the WAVES language
primitives are currently only available via
the WAVES standard package
implementations, and 2) The translation of
the WAVES port list signals into signals
that can be consumed by the VHDL model.

The first issue of simulation efficiency is
expected to be addressed when VHDL tool
vendors implement WAVES in the

available as Unix tar, Unix compressed tar,
or GNU zip tar files. The source code
version comes with a developers guide that
dокументes the code and the tools used to
develop the compilation tool. The
executable version comes with a user's
guide.

Another tool that is being developed at
Rome Laboratory is a WAVES level 1
syntax checker. This tool will allow users
to verify the syntax of any level 1 data set.
This tool is being integrated with the
compilation tool and is planned for release
by the summer of '95. The availability of
these tools will make using WAVES easier
for VHDL model verification.
simulator kernels of their VHDL simulation tools, much like the IEEE 1164 standard has been optimized for simulation by a number of tool vendors.

The second issue may be addressed in a similar manner given the existence of a standard set of WAVES declarations for use with the 1164 standard. A standard set of WAVES declarations would allow for automatic interpretation of the WAVES port list into 1164 signals for VHDL simulation of a WAVES test set with 1164 compliant models. This, or some other optimization technique, would obviate the need to translate every event that occurs on the WAVES port list into signals usable to the model.

In spite of these drawbacks, WAVES provides a powerful mechanism for documenting and applying test sets for VHDL model verification. As more WAVES tools become available, the standard will become easier to use and the benefits of using WAVES will become more obvious. A unified method for documenting test during the design process will hopefully lead to consideration of test issues earlier in the design cycle. As test becomes more important during design, new tools and techniques for design to test will become more prevalent. Perhaps then we will truly realize the promised benefits of concurrent engineering to the fullest.

REFERENCES


