RL-TR-94-76
In-House Report
June 1994

AD-A285 208

WAVES VHDL INTERFACE

James P. Hanna

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

DTIC
ELECTE
OCT 04 1994
G

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

94-31332

94

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-76 has been reviewed and is approved for publication.

APPROVED: Eugene C. Blackburn

EUGENE C. BLACKBURN
Chief, Electronics Reliability Division
Electromagnetics & Reliability Directorate

FOR THE COMMANDER: John J. Bart

JOHN J. BART
Reliability Sciences
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (ERDD) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1994	In-House

4. TITLE AND SUBTITLE
WAVES VHDL INTERFACE

5. FUNDING NUMBERS
PE - 62702F
PR - 2338
TA - 01
WU - 7U

6. AUTHOR(S)
James P. Hanna

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rome Laboratory (ERDD)
525 Brooks Road
Griffiss AFB NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (ERDD)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-76

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: James P. Hanna/ERDD/(315) 330-2241

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Waveform and Vector Exchange Specification (WAVES) is the Industry standard representation for digital stimulus and response for both the design and test communities. The VHSIC Hardware Description Language (VHDL) is the Industry standard language for the design, modeling, and simulation of digital electronics. Together VHDL and WAVES provide powerful support for top-down design and test methodologies and concurrent engineering practices. Although the syntax of WAVES is a subset of VHDL, no special support for using WAVES in a VHDL environment is defined within the language. This report will introduce and describe a VHDL package that was developed at Rome Laboratory to provide a software interface to support the use of WAVES in a VHDL environment. This VHDL package is referred to as the WAVES VHDL interface and has been proposed as a standard practice for a top-down design and test methodology using WAVES and VHDL. This report is not intended to provide a tutorial on VHDL or WAVES. It is assumed that the reader has an adequate understanding of the VHDL language and some modeling techniques. Further, it is assumed that the reader has an understanding of the WAVES language and can follow a simple Level 1 dataset description.

14. SUBJECT TERMS
VHDL, VHSIC, Design, Test, Top Down Design, Design Verification, WAVES, Concurrent Engineering

15. NUMBER OF PAGES 68

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

DTIC QUALITY INSPECTED 5

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Table Of Contents

Abstract 1
Introduction 1
Objective 3
The WAVES-VHDL Interface 4
The Simulator_Codes Package 5
Context Clauses 7
Type Declarations 7
Constant Declarations 8
The WAVES_VHDL_Interface Package 8
Context Clauses 13
Type Declarations 14
Functions 14
Procedures 17
Testbench Example 19
Context Clauses 24
Entity Declaration 25
Architecture Body 25
Conclusions 26
Appendix A: WVI Package Body A1
Appendix B: Four Bit ALU VHDL Model B1
Appendix C: The WAVES Dataset C1
References & Bibliography R1

Abstract

The Waveform and Vector Exchange Specification (WAVES)[1] is the Industry standard representation for digital stimulus and response for both the design and test communities. The VHSIC Hardware Description Language (VHDL)[2] is the Industry standard language for the design, modeling, and simulation of digital electronics. Together VHDL and WAVES provide powerful support for top-down design and test methodologies and concurrent engineering practices. Although the syntax of WAVES is a subset of VHDL, no special support for using WAVES in a VHDL environment is defined within the language. This report will introduce and describe a VHDL package that was developed at Rome Laboratory to provide a software interface to support the use of WAVES in a VHDL environment. This VHDL package is referred to as the WAVES VHDL Interface and has been proposed as a standard practice for a top-down design and test methodology using WAVES and VHDL. This report is not intended to provide a tutorial on VHDL or WAVES. It is assumed that the reader has an adequate understanding of the VHDL language and some modeling techniques. Further, it is assumed that the reader has an understanding of the WAVES language and can follow a simple Level 1 dataset description.

Introduction

In December of 1991, the Waveform and Vector Exchange Specification (WAVES) was approved by the broad balloting community of the Institute of Electrical and Electronics Engineers (IEEE) as the Industry standard representation and exchange format for digital stimulus and response data. The standard (IEEE Std 1029.1-1991) was developed to address the needs of both design and test activities. WAVES was developed by the WAVES Analysis and Standardization Group (WASG). The WASG was jointly sponsored by the Automatic Test Program Generation (ATPG) subcommittee of the Standards Coordination Committee 20 (SCC20) and the Design Automation Standards Subcommittee (DASS) of the Computer Society.

WAVES VHDL Interface

The development of WAVES was funded by the Rome Laboratory Microelectronics Reliability Division as one of the extensions to the Tester Independent Support Software System (TISSS). WAVES was one of the data formats used to demonstrate the capability of automatically generating Automated Test Equipment (ATE) test programs for complex digital Line Replaceable Modules (LRMs) for the Demonstration and Validation (DEMVAL) phase of the Air Force's Advanced Tactical Fighter (ATF) program.

The WAVES standard provides a powerful support mechanism for concurrent engineering practices by allowing digital stimulus and response information to be freely exchanged between multiple simulation and test platforms. WAVES is defined as a syntactic subset of VHDL. However, no "hooks" or interface provisions that are unique to VHDL were designed into the WAVES language to facilitate its use in a VHDL modeling environment. Providing a seamless interface between WAVES datasets and VHDL models (or any specific application) is beyond the scope of developing an interchange format. However, this very lack of interface "hooks" requires the developer of WAVES testbenches for the VHDL environment to have detailed knowledge of the implementation of the WAVES construct known as the WAVES port list.

The interface to the WAVES dataset is through the WAVES port list. The WAVES port list provides the signals that are used to drive the model and signals that describe the expected response of the model. These signals are provided in terms of the WAVES concept of a "logic value." The logic value is described as an event on a given signal in terms of state, strength, direction, and relevance. All of this event information may not be applicable to the VHDL model, some may only have meaning in the context of physical hardware. When WAVES is used in a VHDL modeling environment, the WAVES logic values must be translated into the simulator codes that can be used by the model. No mechanism for achieving this translation is provided by the WAVES language definition since this is a VHDL-specific implementation issue.

The WAVES Language Reference Manual (LRM) does not define the data structure implementation of the WAVES port list. The reason for this is that

particular environments (various testers, various simulators, etc.) may place different constraints and requirements on the underlying implementation of the WAVES port list. The WAVES standard is not dependent on any particular environment or usage scenario. This provides for maximum flexibility when interfacing WAVES to various applications and environments. Flexibility does not come without a price. This poses concern when using WAVES in a VHDL modeling environment since different VHDL simulators are free to implement the WAVES port list in very different ways. Developing WAVES testbenches for VHDL models, then, is not as straight-forward as one might assume. Using WAVES in the VHDL environment is not "plug-and-play."

Objective

The objective of this technical report is to describe and document the work that was done to provide the VHDL community with a user-friendly, seamless interface to WAVES. This work was done in response to several negative comments received during the balloting activities of the WAVES standardization effort. These comments addressed a lack of flexibility in the structure and use of the WAVES port list.

One of the comment ballots returned was concerned that WAVES forces the user to deal with waveforms as a "...flat list of scalar connections..." and that this is "...unreasonable...[for] modern chips and boards with hundreds of scalar connections." This comment further stated that, "It should be easier to identify the connections used in groups and address their waveforms on this basis." Another comment ballot was concerned that, "An undesirable limitation...is...that [WAVES] allows no functional vectors (such as ..., bussed vectors)."

An additional concern was raised during the WASG meeting in which the ballot comments were addressed and responses prepared. The concern was that since the WAVES standard allows flexibility in the implementation of

the WAVES port list that WAVES testbenches would not be portable across multiple implementations.

The WAVES-VHDL Interface (WVI), described in this report, was developed to address these concerns and to serve as a set of "standard practices" for using WAVES in a VHDL modeling environment. The remainder of this technical report discusses these issues. This report presents the Interface packages, describes their functionality, and provides an example of their use in a fully portable VHDL test bench. A "full blown" VHDL 4-bit ALU example, the associated WAVES dataset, and an implementation of the interface package body are included as appendices to this document.

The WAVES-VHDL Interface

The WAVES-VHDL Interface (WVI) was developed to define and provide a software interface to ease the use of WAVES in the VHDL environment. In particular, the WVI addresses two areas of concern: connecting the WAVES port list signals to the VHDL model and test bench portability. If WAVES testbenches are to be portable among a variety of VHDL simulators, without regard to a particular implementation of the WAVES standard, some "standard practice" interface to the WAVES port list must be defined. The WVI provides a nomenclature and a set of calling conventions for an implementation independent functional interface to the WAVES port list. These functional interfaces also provide the mechanisms needed to support the WAVES port list to VHDL model port connections in a flexible and easy to use manner. The functions and procedures included in the WVI package provide support for comparing the actual response of the VHDL model with the expected response given in the WAVES dataset.

The WVI uses the VHDL packaging concepts to separate the functional interface from the implementation details. The WVI package specification serves as the "standard practice" that defines these interfaces and their calling conventions. The WVI package body defines the semantics behind the interfaces and provides one possible implementation of these semantics

based on the VHDL packages that implement the WAVES standard. The package body, listed in Appendix A, is only one possible implementation that demonstrates these semantics. This approach allows testbenches written based on the WVI to be portable among multiple WAVES-VHDL environments since they rely only on the WVI package specification and not on the implementation hidden in the package body. VHDL environments with differing WAVES implementations will need to provide a WVI package body based on the local WAVES implementation. Since the implementation of the WVI package body is only dependent on the implementation of the WAVES port list, modifying these functions and procedures is a relatively trivial task.

The WVI also provides the needed WAVES logic value to VHDL simulator code translation mechanisms. The WVI "uses" a user supplied VHDL package that defines the mappings between the WAVES logic values and the simulator codes that the VHDL model expects and generates. This user supplied package allows the WVI to support VHDL models with arbitrary signal types.

The Simulator_Codes Package

A user supplied package that must be named "Simulator_Codes" is required in order to use the WVI package. This package defines the explicit mappings between the simulator codes that the VHDL model expects and the logic values that the WAVES dataset provides. This is the only code that the user of the WVI must supply in order to use the WAVES_VHDL_Interface package, presented later.

This package consists only of subtype declarations, type declarations, and constant declarations. The types declared within this package are used only in the definition of the constants that this package declares. Listing 1 provides an example of a Simulator_Codes package that maps the simulation codes from IEEE STD 1164 STD_ulogic type to the WAVES example dataset that is described later in this report.

Listing 1: The Simulator_Codes Package.

```
use WORK.STD_Logic_1164.all;
use WORK.MM54HC181_WAVES_logic.all;
package Simulator_Codes is
 --
 -- Declare Sim_code.
 --
 subtype Sim_code is Std_ulogic;
 --
 -- Declare Sim_code_vector.
 --
 subtype Sim_code_vector is Std_ulogic_vector;
 --
 -- Declare Sim_code_array.
 --
 type Sim_code_array is array ( Logic_value'left to Logic_value'right )
                of Sim_code;
 --
 -- Declare Boolean_matrix.
 --
 type Boolean_matrix is array ( Sim_code'left to Sim_code'right,
                                Logic_value'left to Logic_value'right )
                of Boolean;
 --
 -- Define TRANSLATE table.
 --
 -- LV_UNKNOWN  DONT_CARE  DRIVE_0  DRIVE_1  SENSE_0  SENSE_1
 --
 Constant TRANSLATE : Sim_code_array := ( 'U','-', '0', '1', '0', '1' );
 --
 -- Define IS_EQUAL table.
 --
 constant IS_EQUAL : Boolean_matrix :=
 --
```

```
-- LV_UNKNOWN DONT_CARE DRIVE_0 DRIVE_1 SENSE_0 SENSE_1
--
(( TRUE,    TRUE,   FALSE,  FALSE,  FALSE,  FALSE ), -- 'U'
 ( TRUE,    TRUE,   FALSE,  FALSE,  FALSE,  FALSE ), -- 'X'
 ( FALSE,   TRUE,   TRUE,   FALSE,  TRUE,   FALSE ), -- '0'
 ( FALSE,   TRUE,   FALSE,  TRUE,   FALSE,  TRUE  ), -- '1'
 ( FALSE,   TRUE,   FALSE,  FALSE,  FALSE,  FALSE ), -- 'Z'
 ( TRUE,    TRUE,   FALSE,  FALSE,  FALSE,  FALSE ), -- 'W'
 ( FALSE,   TRUE,   TRUE,   FALSE,  TRUE,   FALSE ), -- 'L'
 ( FALSE,   TRUE,   FALSE,  TRUE,   FALSE,  TRUE  ), -- 'H'
 ( TRUE,    TRUE,   TRUE,   TRUE,   TRUE,   TRUE  )); -- '-'

end Simulator_Codes;
```

Context Clauses

The "use WORK.STD_Logic_1164.all" clause in Listing 1 provides visibility to all of the types and functionality of the 1164 standard packages.

The "use WORK.MM54HC181_WAVES_logic.all" in Listing 1 provides visibility to the required, user-defined WAVES type Logic_value.

Type Declarations

Two subtypes and two constants are exported by this package and used by the WVI package. The subtypes must be named "Sim_code" and "Sim_code_vector." The subtype "Sim_code", in this example, essentially renames the IEEE STD 1164 STD_ulogic type for use with the WAVES_VHDL_Interface package. Likewise, the subtype "Sim_code_vector", in this example, renames the IEEE STD 1164 STD_ulogic type for use with the WAVES_VHDL_Interface package.

The two types, "Sim_code_array" and "Boolean_matrix", serve only to provide the data structures necessary to implement the two constant look-up tables

"TRANSLATE" and "IS_EQUAL" respectively. The name of these two types is irrelevant since they are not used anywhere else.

Constant Declarations

Two constants are used to implement look-up tables. These constants must be named "TRANSLATE" and "IS_EQUAL." The constant "TRANSLATE," as its name implies, is used by the WAVES_VHDL_Interface package to translate WAVES logic values into the appropriate simulator codes for use with the VHDL model. This look-up table is indexed by the WAVES type Logic_value and must correspond exactly in order and dimension with the WAVES Logic_value declaration. The semantics of the "TRANSLATE" table are as follows: given a WAVES Logic_value, TRANSLATE returns the corresponding Sim_Code.

The constant "IS_EQUAL" is used by the WAVES_VHDL_Interface package to determine the equivalence of a WAVES Logic_value and a Sim_code. The semantics of the "IS_EQUAL" table are as follows: given a Sim_code and a WAVES Logic_value, IS_EQUAL returns TRUE if, and only if, the Sim_code and Logic_value are logically equivalent. In this example the WAVES Logic_value 'SENSE_0' and the IEEE STD Ulogic value '-' are defined as logically equivalent since the corresponding table entry is the Boolean 'TRUE'.

The WAVES VHDL Interface Package

As stated previously, the WAVES VHDL Interface package provides a nomenclature and a set of calling conventions for an implementation independent functional interface to the WAVES port list. This package consists of one type declaration, eight function declarations, and seven procedure declarations. In this section each of these functional interfaces will be described. Listing 2 lists the WVI package specification in its entirety.

Listing 2: The WAVES_VHDL_Interface Package Specification.

```
use WORK.MM54HC181_WAVES_DUT.all;
use WORK.MM54HC181_WAVES_Logic.all;
use WORK.Simulator_Codes.all;
use WORK.WAVES_Objects.all;
package WAVES_VHDL_Interface is

--
-- Type declarations.
--
  --
  -- Declare a type for vectors of user defined type 'Logic_value'.
  --
  type Logic_vector is array ( natural range <> ) of Logic_value;


--
-- Function and procedure declarations.
--
  --
  -- This function returns the Sim_code that corresponds to the WAVES
  -- Logic_value of the given pin in the waves port list.
  --
  function sim_code_Of( signal WPL : WAVES_port_list;
                                PIN  : Test_pins )
                return Sim_code;

  --
  -- This function returns a Sim_code_vector that corresponds - in length
  -- and order - to the WAVES Logic values in the WAVES port list elements
  -- that are members of the given pin set.
  --
  function sim_code_Vector_Of( signal WPL          : WAVES_port_list;
                                      PIN_SET      : Pinset := ALL_PINS;
```

```
                                        ASCENDING : Boolean := TRUE )
        return Sim_code_vector;

--
-- This function returns the WAVES Logic_value of the given pin in
-- the waves port list.
--
function Logic_Value_Of( signal WPL : WAVES_port_list;
                                  PIN  : Test_Pins )
        return Logic_value;

--
-- This function returns a Logic_vector that corresponds - in length
-- and order - to WAVES logic values in the WAVES port list elements
-- that are members of the given pin set.
--
function Logic_Vector_Of( signal WPL          : WAVES_port_list;
                                  PIN_SET     : Pinset := ALL_PINS;
                                  ASCENDING : Boolean := TRUE )
        return Logic_vector;

--
-- This function compares a Sim_code to the corresponding element
-- (given by PIN) of the WAVES port list.
--
function Compare( signal WPL  : WAVES_port_list;
                          CODE : Sim_code;
                          PIN  : Test_pins )
        return Boolean;

--
-- This function compares the Sim_codes in the Sim_code_vector to the
-- corresponding elements (given by PIN_SET) of the WAVES port list.
--
function Compare( signal WPL    : WAVES_port_list;
                          VECTOR : Sim_code_vector;
```

```
                                PIN_SET : Pinset := ALL_PINS )
        return Boolean;

--
-- This function compares a Sim_code to an element (given by PIN) of
-- the WAVES port list and returns a WAVES_match_list indicating the result.
--
function Compare( signal WPL    : WAVES_port_list;
                                CODE : Sim_code;
                                PIN     : Test_pins )
        return WAVES_match_list;

--
-- This function compares each Sim_code in the Sim_code_vector to the
-- corresponding elements (given by PIN_SET) of the given WAVES port
-- list and returns a WAVES_match_list indicating the results.
--
function Compare( signal WPL        : WAVES_port_list;
                                VECTOR  : Sim_code_vector;
                                PIN_SET : Pinset := ALL_PINS )
        return WAVES_match_list;

--
-- This procedure assigns the element (given by PIN) of the WAVES
-- match list to the values specified in the WAVES_match_list that are
-- given by PIN.
--
procedure Match_Assign( signal WML     : inout WAVES_match_list;
                                        VALUE : in     WAVES_match_list;
                                        PIN       : in     Test_Pins );

--
-- This procedure assigns all of the elements (given by PIN_SET) of
-- the WAVES match list to the values specified in the WAVES_match_list
-- that are members of PIN_SET.
--
```

```
procedure Match_Assign( signal WML       : inout WAVES_match_list;
                                  VALUE    : in    WAVES_match_list;
                                  PIN_SET : in    Pinset := ALL_PINS );

--
-- This procedure compares a Sim_code to an element (given by PIN) of
-- the WAVES port list and assigns the corresponding element of the
-- WAVES match list to the result.
--
procedure Compare( signal WPL    : inout WAVES_port_list;
                          signal WML   : inout WAVES_match_list;
                                 CODE : in     Sim_code;
                                 PIN    : in     Test_pins );

--
-- This procedure compares each Sim_code in the Sim_vector to the
-- elements (given by PIN_SET) of the WAVES port list and assigns
-- the corresponding elements of the WAVES match list to the results.
--
procedure Compare( signal WPL       : inout WAVES_port_list;
                          signal WML       : inout WAVES_match_list;
                                 VECTOR  : in     Sim_code_vector;
                                 PIN_SET : in     Pinset := ALL_PINS );

--
-- This procedure returns the tag string associated with the given
-- pin from the WAVES port list.
--
procedure Get_Tag( signal WPL : inout WAVES_port_List;
                                 TAG :    out String;
                                 PIN  : in      Test_pins );

--
-- This procedure returns the tag string associated with the given
-- pin set from the WAVES port list.
--
```

```
    procedure Get_Tag( signal WPL      : inout WAVES_port_List;
                                TAG      :    out String;
                                PIN_SET : in      Pinset := ALL_PINS );

    --
    -- This procedure returns the tag string associated with the current
    -- WAVES port list.
    --
    procedure Get_Tag( signal WPL : inout WAVES_port_List;
                                TAG :    out String );

end WAVES_VHDL_Interface;
```

Context Clauses

The "use WORK.MM54HC181_WAVES_DUT.all" clause in Listing 2 is required to provide visibility to the user defined WAVES required "Test_pins" type declaration. This type declaration is used by the WAVES VHDL Interface package functions and procedures to allow the user to select signals from the WAVES port list by pin name.

The "use WORK.MM54HC181_WAVES_Logic.all" clause in Listing 2 is required to provide visibility to the user defined WAVES required Logic_value type declaration. This type declaration is used by the WAVES VHDL Interface package to allow the translation of logic values to Simulator codes and for function return types.

The "use WORK.Simulator_Codes.all" clause is required to provide visibility to the types and translation tables declared in the Simulator_Codes package. The types are Sim_code and Sim_code_vector and are used as function return types. The translation tables are TRANSLATE and IS_EQUAL and are used to compute Logic_value to Sim_code translations and Logic_value to Sim_code logical equivalence respectively.

Finally, the "use WORK.WAVES_Objects.all" clause is required to provide visibility to the WAVES built-in types Pinset, WAVES_port_list, and WAVES_match_list. The type Pinset is used to allow the user to select signals from the WAVES port list by pinset (logical pin groupings). The type WAVES_port_list is the data structure from which individual signals and groups of signals are selected for use by the VHDL model. The WAVES_match_list type is used as function and procedure return type.

Type Declarations

The type declaration, Logic_vector is used as the return type for the Logic_Vector_Of function. This function is discussed below.

Functions

The function **Sim_Code_Of** accepts two parameters: WPL - of type WAVES_port_list and PIN - of type Test_pins, and returns an object of type Sim_code. The value returned is the simulator code (defined by the translation table TRANSLATE in the Simulator_Codes package) that corresponds to the logic value located in the *PINth* element of the WAVES port list object WPL. This function is used to select the Sim_code of a single signal from the WAVES port list. The result of a call to this function may be applied to an input port of a VHDL model and used to stimulate the model during simulation.

The function **Sim_Code_Vector_Of** accepts three parameters: WPL - of type WAVES_port_list, PIN_SET - of type Pinset, and ASCENDING - of type Boolean, and returns an object of type Sim_code_vector. The value returned is the vector of simulator codes that correspond to the logic values located in the WAVES port list object WPL and indexed by the elements of the Pinset object PIN_SET. Both the PIN_SET and the ASCENDING parameters have default values. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The default value of the ASCENDING parameter is the Boolean TRUE. The semantics of the function are as follows: for each element in PIN_SET select the single corresponding logic value from the WAVES port list, then translate (as

defined by the translation table TRANSLATE in the Simulator_Codes package) the selected logic value into a simulator code and place it in the *ith* element of the returned vector. The base type of the range of the returned vector is Natural (bounded by 0), and, is in ascending order when the Boolean value of the ASCENDING parameter is TRUE. This function is used to select a vector of Sim_code elements (eg. a bus) with either ascending or descending range from the WAVES port list. The result of a call to this function may be applied to an input port (or ports) of a VHDL model and used to stimulate the model during simulation.

The function **Logic_Value_Of** accepts two parameters: WPL - of type WAVES_port_list and PIN - of type Test_pins, and returns an object of type Logic_value. The value returned is the Logic_value located in the *PINth* element of the WAVES port list object WPL. This function is used to select the Logic_value of a single signal from the WAVES port list.

The function **Logic_Vector_Of** accepts three parameters: WPL - of type WAVES_port_list, PIN_SET - of type Pinset, and ASCENDING - of type Boolean, and returns an object of type Logic_vector. The value returned is the vector of logic values located in the WAVES port list object WPL and indexed by the elements of the Pinset object PIN_SET. Both the PIN_SET and the ASCENDING parameters have default values. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The default value of the ASCENDING parameter is the Boolean TRUE. The semantics of the function are as follows: for each element in PIN_SET select the single corresponding logic value from the WAVES port list and place it in the *ith* element of the returned vector. The base type of the range of the returned vector is Natural (bounded by 0), and, is in ascending order when the Boolean value of the ASCENDING parameter is TRUE. This function is used to select a vector of Logic_value elements (e.g. a bus) with either ascending or descending range from the WAVES port list.

There are four overloaded compare functions grouped into two variants based on their return types. One variant of the compare function returns a Boolean value while the other returns a WAVES_match_list value. The first variant

is used to compare simulator codes that are generated by a VHDL model to the expected, or predicted, response specified by the WAVES dataset. The second variant accomplished the same comparison but instead of returning a Boolean value, this comparison is used to generate a WAVES match list object that can be passed to the waveform generator procedure to take advantage of the match capability of WAVES. Each of the compare functions are discussed below.

The first **Compare** function (first variant) accepts three parameters: WPL - of type WAVES_port_list, CODE - of type Sim_code, and PIN - of type Test_pins, and returns an object of type Boolean. The semantics of this function are as follows: select the logic value located in the WAVES port list parameter WPL at the *PINth* location, and compare it to the simulator code parameter CODE (based on the look-up table IS_EQUAL in the Simulator_Codes package) then return the Boolean value that results from the table look-up. This function is used to compare a single simulator code with a single signal in the WAVES port list. The result of a call to this function may be used to monitor the behavior of output ports of a VHDL model during simulation.

The next **Compare** function (first variant) accepts three parameters: WPL - of type WAVES_port_list, VECTOR - of type Sim_code_vector, and PIN_SET - of type Pinset, and returns an object of type Boolean. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The semantics of this function are as follows: for each element in PIN_SET select the single corresponding logic value located in the WAVES port list parameter WPL and compare it to the *ith* element of the Sim_code_vector parameter VECTOR (based on the look-up table IS_EQUAL in the Simulator_Codes package). The return value of the function is the logical and of all of the individual comparisons. This function is used to compare a vector of simulator codes (e.g. a bus) with a vector of logic value signals in the WAVES port list. The result of a call to this function may be used to monitor the behavior of output ports of a VHDL model during simulation.

The next **Compare** function (second variant) accepts three parameters: WPL - of type WAVES_port_list, CODE - of type Sim_code, and PIN - of type

Test_pins, and returns an object of type WAVES_match_list. The semantics of this function are as follows: examine the match control flag for the *PINth* element of the WAVES port list parameter WPL. If the value of the match control flag is "SAMPLE," select the logic value located in the WAVES port list parameter WPL at the *PINth* location, and compare it to the simulator code parameter CODE (based on the look-up table IS_EQUAL in the Simulator_Codes package). Then store the result of this comparison in the match result register of the *PINth* element of the resulting WAVES_match_list. This function is used to compare a single simulator code with a single Logic_value signal in the WAVES port list and generate a WAVES_match_list that represents this comparison.

The last **Compare** function (second variant) accepts three parameters: WPL - of type WAVES_port_list, VECTOR - of type Sim_code_vector, and PIN_SET - of type Pinset, and returns an object of type WAVES_match_list. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The semantics of this function are the same as the other function of this variant except that instead of generating a WAVES match list with only a single pin comparison represented, this function generates a WAVES match list that represents the comparisons for multiple pins, all of the elements in the PIN_SET parameter. This function is used to compare a vector of simulator codes (e.g. a bus) with a vector of Logic_value signals in the WAVES port list and generate a WAVES_match_list that represents this comparison.

Procedures

The first of two overloaded **Match_Assign** procedures accepts three parameters: WML - of type WAVES_match_list, VALUE - of type WAVES_match_list, and PIN - of type Test_pins. The result of calling this procedure is to assign the *PINth* element of the WAVES match list parameter WML, the value stored in the *PINth* element of the WAVES match list parameter, VALUE.

The second overloaded **Match_Assign** procedure accepts three parameters: WML - of type WAVES_match_list, VALUE - of type WAVES_match_list,

and PIN_SET - of type Pinset. The default value of the PIN_SET parameter is the set ALI _PINS, as defined in the WAVES standard package WAVES_Objects. The result of calling this procedure is to assign the elements of the WAVES match list parameter WML, the value stored in the elements of the WAVES match list parameter, VALUE that are selected by each pin in PIN_SET.

The first of two overloaded **Compare** procedures accepts four parameters: WPL - of type WAVES_port_list, WML - of type WAVES_match_list, CODE - of type Sim_code, and PIN - of type Test_pins. The result of calling this procedure is equivalent to calling the single pin version of the Compare function (second variant) and passing the resulting WAVES match list to the single pin version of the Match_Assign procedure. This procedure performs all of the functions of these two subprograms.

The second overloaded **Compare** procedure accepts four parameters: WPL - of type WAVES_port_list, WML - of type WAVES_match_list, VECTOR - of type Sim_code_vector, and PIN_SET - of type Pinset. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The result of calling this procedure is equivalent to calling the vector version of the Compare function (second variant) and passing the resulting WAVES match list to the vector version of the Match_Assign procedure. This procedure performs all of the function of these two subprograms.

The final three procedures in the WAVES_VHDL_Interface package are overloaded Get_Tag procedures. The purpose of these procedures is to extract the tag strings that may have been assigned to the waveform.

The first **Get_Tag** procedure accepts three parameters: WPL - of type WAVES_port_list, TAG - of type String, and PIN - of type Test_pins. The result of calling this procedure is to assign the TAG parameter the value of the tag string associated with the *PINth* element of the WAVES port list parameter WPL.

The next **Get_Tag** procedure accepts three parameters: WPL - of type WAVES_port_list, TAG - of type String, and PIN_SET. The default value of the PIN_SET parameter is the set ALL_PINS, as defined in the WAVES standard package WAVES_Objects. The result of calling this procedure is to assign the TAG parameter the value of the tag string associated with each element of the WAVES port list parameter WPL that is selected by the elements in PIN_SET.

The last **Get_Tag** procedure accepts two parameters: WPL - of type WAVES_port_list, and TAG - of type String. The result of calling this procedure is to assign the TAG parameter the value of the tag string associated with the entire slice of the waveform represented by the WAVES port list parameter WPL.

Testbench Example

Listing 3 lists the code for an example test bench that illustrates some of the functionality of the WAVES VHDL Interface package. This example demonstrates the use of the Sim_Code_Of, Sim_Code_Vector_Of, and the Compare functions. This example test bench verifies the functionality of a VHDL model of a four bit ALU using a WAVES Level 1 dataset. The source code of the four bit ALU model (including brief descriptive text) is given in Appendix B. The source code of the WAVES Level 1 dataset (including brief descriptive text) is listed in Appendix C. The test bench example is described below.

Listing 3: The VHDL Testbench Example.

```
--------------------------------------------------------------------------
--
-- This design unit provides a functional test bench for verifying the
-- 181 ALU using a WAVES dataset.
```

```
--
------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_Logic_1164.all;
use WORK.MM54HC181_WAVES_DUT.all;
use WORK.WAVES_Objects.all;
use WORK.WAVES_VHDL_Interface.all;
use WORK.Waveform_Generator.all;
use WORK.MM54HC181_CMOS_PERFORMANCE_CHARACTERISTICS.all;

entity Test_Bench is

 generic ( TEST_VOLTAGE       : natural := 1;
           TEST_TEMPERATURE : natural := 0 );

end Test_Bench;

architecture ALU of Test_Bench is

 --
 -- Define signals to connect the WAVES port list signals to the DUT inputs.
 --
 signal APINS : std_ulogic_vector( 3 downto 0 );
 signal BPINS : std_ulogic_vector( 3 downto 0 );
 signal SPINS : std_ulogic_vector( 3 downto 0 );
 signal CARRY : std_ulogic;
 signal MODE  : std_ulogic;

 --
 -- Define a signal to receive the F pin outputs from the DUT.
 --
 signal FUNCTION_OUTPUTS : Std_ulogic_vector( 0 to 3 );

 --
 -- Define a signal to receive the AEQB pin outputs from the DUT.
```

```
--
signal A_EQUAL_B : Std_ulogic;

--
-- Define a signal to receive the CN_4 pin outputs from the DUT.
--
signal CN4 : Std_ulogic;

--
-- Define a signal to receive the NOTG pin outputs from the DUT.
--
signal NOT_G : Std_ulogic;

--
-- Define a signal to receive the NOTP pin outputs from the DUT.
--
signal NOT_P : Std_ulogic;

--
-- Define a signal to receive each slice of the WAVES dataset
--
signal CONNECT : WAVES_port_list;

--
-- The DUT Component Declaration.
--
component FOUR_BIT_ALU
 generic ( VOLTAGE : Volts;
           OPERATING_TEMPERATURE : Temperature );
 port ( A_INPUTS, B_INPUTS, SELECT_LINES : in Std_ulogic_vector;
        Cn, M : in Std_ulogic;
        F_OUTPUTS : out Std_ulogic_vector;
        AEQB, CN_4, NOTG, NOTP : out Std_ulogic );
end component;

--
```

```
 -- Configure the DUT.
 --
 for ALU : FOUR_BIT_ALU use entity WORK.MM54HC181( BEHAVIORAL );

begin

 --
 -- Process to generate the waveform.
 --
 WAVES : Waveform( CONNECT );


 XTRACT : process( CONNECT )
   begin
     APINS <=   Sim_Code_Vector_Of( CONNECT, A_PINS );
     BPINS <=   Sim_Code_Vector_Of( CONNECT, B_PINS );
     SPINS <=  Sim_Code_Vector_Of( CONNECT, S_PINS );
     CARRY <= Sim_Code_Of( CONNECT, CN );
     MODE  <= Sim_Code_Of( CONNECT, M );
   end process;

 --
 -- Connect the DUT model.
 --
 ALU : FOUR_BIT_ALU
   generic map ( VOLTAGE =>
                       Volts'val( TEST_VOLTAGE ),
                 OPERATING_TEMPERATURE =>
                       Temperature'val( TEST_TEMPERATURE ) )

   port map ( A_INPUTS       => APINS,
              B_INPUTS       => BPINS,
              SELECT_LINES => SPINS,
             CN              => CARRY,
             M               => MODE,
              F_OUTPUTS      => FUNCTION_OUTPUTS,
```

```
        AEQB        => A_EQUAL_B,
        NOTP        => NOT_P,
        CN_4        => CN4,
        NOTG        => NOT_G );


MONITOR_FUNCTION_OUTPUTS:
   process( CONNECT, FUNCTION_OUTPUTS )
   begin
      assert ( Compare( CONNECT, FUNCTION_OUTPUTS, F_PINS ) )
      report "==> Error in function outputs."
      severity WARNING;
   end process;

MONITOR_COMPARATOR:
   process( CONNECT, A_EQUAL_B )
   begin
      assert ( Compare( CONNECT, A_EQUAL_B, AEQB ) )
      report "==> Error in comparator output."
      severity WARNING;
   end process;

MONITOR_CARRY_PROPOGATE:
   process( CONNECT, NOT_P )
   begin
      assert ( Compare( CONNECT, NOT_P, NOTP ) )
      report "==> Error in carry propagate output."
      severity WARNING;
   end process;

MONITOR_CARRY:
   process( CONNECT, CN4 )
   begin
      assert ( Compare( CONNECT, CN4, CN_4 ) )
      report "==> Error in carry output."
      severity WARNING;
```

```
    end process;

  MONITOR_CARRY_GENERATE:
    process( CONNECT, NOT_G )
    begin
      assert ( Compare( CONNECT, NOT_G, NOTG ) )
      report "==> Error in carry generate output."
      severity WARNING;
    end process;
end ALU;
```

Context Clauses

The "use IEEE.STD_Logic_1164.all" clause in Listing 3 is required to provide visibility to all of the types and functionality of the IEEE Standard 1164 packages.

The "use WORK.MM54HC181_WAVES_DUT.all" clause is required to provide visibility to the required, user defined WAVES type Test_pins.

The "use WORK.WAVES_Objects.all" is required to provide visibility to the WAVES type WAVES_port_list.

The "use WORK.WAVES_VHDL_Interface.all" clause is required to provide visibility to all of the types and functionality of the WAVES_VHDL_Interface package.

The "use WORK.Waveform_Generator.all" clause is required to provide visibility to the WAVES waveform generator procedure "Waveform." This procedure is used to supply the WAVES signals to the VHDL model one slice at a time.

The "use WORK.MM54HC181_CMOS_Performance_Characteristics.all" clause is required to provide visibility to the types Volts, and Temperature used in the generic clause of the "Test_bench" entity declaration.

Entity Declaration

The VHDL entity declaration "Test_Bench" declares two generics: TEST_VOLTAGE and TEST_TEMPERATURE. These generics are used to allow the test bench to alter the operating conditions under which the model is exercised.

Architecture Body

The architecture body of the test bench first declares all of the signals necessary to connect the WAVES dataset to the four bit ALU model. The next declaration in the test bench is the four bit ALU component declaration. This declaration describes the interface characteristics of the four bit ALU model component that will be used by the test bench. Next, the configuration declaration is used to instantiate the actual ALU component from the library for this test bench. Here the component MM54HC181(BEHAVIORAL) is selected. This is the behavioral model of the four bit ALU that is listed in Appendix B.

The executable portion of the architecture body consists of eight processes. The first process: WAVES, is a VHDL concurrent procedure call. This process calls the procedure Waveform from the waves dataset and returns one slice of the WAVES data through the WAVES port list signal CONNECT. The execution timing, or firing, of this procedure is controlled by the WAVES dataset. The external file, Listing C8 of Appendix C, indicates that one slice is generated every 500 ns.

The second process, XTRACT, is used to extract the input signals from the WAVES signal CONNECT. This process is sensitive on the WAVES port list signal CONNECT. Whenever any event occurs in the CONNECT signal, this process becomes active and calls the WAVES VHDL Interface functions Sim_Code_Of and Sim_Code_Vector_Of to provide values for the signals APINS, BPINS, SPINS, CARRY, and MODE. These are the intermediate signals used to connect the WAVES dataset to the four bit ALU model.

The third process, ALU, provides the mappings from the test bench entity declaration generics (TEST_VOLTAGE and TEST_TEMPERATURE) to the values of the ALU component generics (VOLTAGE and OPERATING_TEMPERATURE) as well as the port mappings from the intermediate signals and the four bit ALU model. This is where the stimuli and response signals are "wired" to the instantiated ALU component.

The following five processes monitor the outputs of the model during the entire simulation. Each process is sensitive to the WAVES port list signal CONNECT so that any event in the CONNECT signal will cause these processes to become active. Each of the five monitor processes are also sensitive on a different output signal of the model. The five processes continuously monitor the model outputs: FUNCTION_OUTPUTS, A_EQUAL_B (comparator), NOT_P (carry propagate), CN4 (Carry), and NOT_G (carry generate). This is necessary to verify that the model outputs correspond to the values specified in the WAVES dataset for the entire simulation.

The assert clauses of the monitor processes use the variant of the Compare function that returns a Boolean value. This function compares the appropriate signals in the CONNECT signal with the signals generated by the model. The assert statement will report an error whenever these signals are not equivalent. Two types of Compare function are demonstrated here: one that selects a single signal from the CONNECT parameter (eg. MONITOR_CARRY), and one that selects a vector of signals from the CONNECT parameter (MONITOR_FUNCTION_OUTPUTS).

Conclusions

Although the WAVES VHDL Interface package provides solutions to many concerns confronting the user of WAVES in a VHDL environment, some concerns still exist. For example, the need to place the entire WAVES port list signal CONNECT in the sensitivity list of each monitor process causes

the process to become active whenever any event on any subsignal of CONNECT occurs. This causes the monitor processes to "fire" many more times than is necessary to verify the model outputs with which each individual monitor process is concerned. This causes the simulation to become inefficient, and, with large pin count devices could make simulation time unreasonably (or impossibly) long. This problem can not be rectified by placing a call to one of the WVI functions to select a subset of signals from CONNECT since functions return values, not signals. Only signals may appear in a process sensitivity list.

The WVI does, however, address all of the concerns outlined in this report. The WVI provides an easy to use set of procedures and functions that hide the details of the WAVES port list implementation from the VHDL modeler that wants to use the WAVES standard to verify his models. Also, using this set of recommended practices assures the portability of the testbenches across multiple WAVES implementations since the WAVES port list implementation details remain hidden from the test bench.

Finally, the concerns raised during comment ballot resolution regarding the ability to deal with sets of pins, or busses, are addressed by the functions and procedures that allow the selection of vectors of signals from the WAVES port list. The use of pin names and pin set names as the signal selectors of these functions and procedures allows the user to specify the signals of interest in a manner that is convenient and familiar.

Appendix A: WVI Package Body

Listing A1 of this Appendix contains the body of the WAVES_VHDL_Interface package. This source code is intended only to illustrate the semantics of the functions and procedures listed in Listing 2. This code illustrates only one possible implementation of these semantics based on the single existing implementation of the WAVES standard packages. Further, this code has not been exhaustively tested to assure that all of its functionality is correct and/or complete.

Listing A1: The WAVES VHDL Interface Package Body.

```
library WAVES_STD;
use WAVES_STD.WAVES_SYSTEM.all;
package body WAVES_VHDL_Interface is

  --
  -- Function to compute the number of elements in the given pin set.
  --
  function Size_Of( PIN_SET : Pinset ) return Natural is

  variable N : Natural := 0;

  begin
    for I in PIN_SET'range loop
      if PIN_SET( I ) then
        N := N + 1;
      end if;
    end loop;
    return N;
  end Size_Of;


  --
  -- Function to create a Sim_code_vector with range n downto 0 from
```

```
-- a Sim_code_vector with range 0 to n.
--
function Reverse_Vector( VECTOR : in Sim_code_vector )
          return Sim_code_vector is

variable R_VECTOR : Sim_code_vector( VECTOR'reverse_range );
variable J        : integer;

begin
   J := VECTOR'length - 1;
   for I in VECTOR'range loop
      R_VECTOR( J ) := VECTOR( I );
      J := J - 1;
   end loop;
   return R_VECTOR;
end Reverse_Vector;



--
-- Function to create a Logic_vector with range n downto 0 from
-- a Logic_vector with range 0 to n.
--
function Reverse_Vector( VECTOR : in Logic_vector )
          return Logic_vector is

variable R_VECTOR : Logic_vector( VECTOR'reverse_range );
variable J        : integer;

begin
   J := VECTOR'length - 1;
   for I in VECTOR'range loop
      R_VECTOR( J ) := VECTOR( I );
      J := J - 1;
   end loop;
   return R_VECTOR;
end Reverse_Vector;
```

```
--
-- This function returns the Sim_code that corresponds to the WAVES
-- Logic_value of the given pin in the waves port list.
--
function sim_code_Of( signal WPL : WAVES_port_list;
                      PIN : Test_pins )
         return Sim_code is

begin
   return TRANSLATE(
        Logic_value'val( WPL.WPL( Test_pins'pos( PIN ) + 1 ).L_VALUE ));
end sim_code_Of;



--
-- This function returns a Sim_code_vector that corresponds - in length
-- and order - to the WAVES Logic values in the WAVES port list elements
-- that are members of the given pin set.
--
function sim_code_Vector_Of( signal WPL      : WAVES_port_list;
                             PIN_SET   : Pinset := ALL_PINS;
                             ASCENDING : Boolean := TRUE )
         return Sim_code_vector is

constant LENGTH : Natural := Size_Of( PIN_SET );
variable VECTOR : Sim_code_vector( 0 to LENGTH - 1 );
variable N      : Natural := 0;

begin
   for PIN in PIN_SET'range loop
     if PIN_SET( PIN ) then
        VECTOR( N ) := sim_code_Of( WPL, PIN );
        N := N + 1;
     end if;
   end loop;
```

```
    if not ASCENDING then
      return Reverse_Vector( VECTOR );
    else
      return VECTOR;
    end if;
  end sim_code_Vector_Of;


  --
  -- This function returns the WAVES Logic_value of the given pin in
  -- the waves port list.
  --
  function Logic_Value_Of( signal WPL : WAVES_port_list;
                           PIN : Test_pins )
           return Logic_value is

  begin
    return Logic_value'val( WPL.WPL( Test_pins'pos( PIN ) + 1 ).L_VALUE );
  end Logic_Value_Of;


  --
  -- This function returns a Logic_vector that corresponds - in length
  -- and order - to WAVES logic values in the WAVES port list elements
  -- that are members of the given pin set.
  --
  function Logic_Vector_Of( signal WPL      : WAVES_port_list;
                            PIN_SET   : Pinset := ALL_PINS;
                            ASCENDING : Boolean := TRUE )
           return Logic_vector is

  constant LENGTH : Natural := Size_Of( PIN_SET );
  variable VECTOR : Logic_vector( 0 to LENGTH - 1 );
  variable N      : Natural := 0;

  begin
```

```
    for PIN in PIN_SET'range loop
      if PIN_SET( PIN ) then
        VECTOR( N ) := Logic_Value_Of( WPL, PIN );
        N := N + 1;
      end if;
    end loop;
    if not ASCENDING then
      return Reverse_Vector( VECTOR );
    else
      retum VECTOR;
    end if;
  end Logic_Vector_Of;

--
-- This function compares a Sim_code to the corresponding element
-- (given by PIN) of the WAVES port list.
--
function Compare( signal WPL  : WAVES_port_list;
                  CODE : Sim_code;
                  PIN  : Test_pins )
         return Boolean is

begin
   return IS_EQUAL( CODE, Logic_value'val(
                    WPL.WPL( Test_pins'pos( PIN ) + 1 ).L_VALUE ));
end Compare;

--
-- This function compares the Sim_codes in the Sim_code_vector to the
-- corresponding elements (given by PIN_SET) of the WAVES port list.
--
function Compare( signal WPL     : WAVES_port_list;
                  VECTOR  : Sim_code_vector;
                  PIN_SET : Pinset := ALL_PINS )
         return Boolean is
```

```
  constant ASCENDING : Boolean := VECTOR'left = VECTOR'low;
  variable RESULT    : Boolean := TRUE;
  variable I         : natural := VECTOR'left;

  begin
    for PIN in Pinset'range loop
      if PIN_SET( PIN ) then
        RESULT := RESULT and Compare( WPL, VECTOR( I ), PIN );
        if ASCENDING then
          I := I + 1;
        else
          I := I - 1;
        end if;
      end if;
    end loop;
    return RESULT;
  end Compare;

  --
  -- This function compares a Sim_code to an element (given by PIN) of
  -- the WAVES port list and returns a WAVES_match_list indicating the result.
  --
  function Compare( signal WPL  : WAVES_port_list;
                    CODE : Sim_code;
                    PIN  : Test_pins )
           return WAVES_match_list is

  constant I      : natural := Test_pins'pos( PIN ) + 1;
  variable RESULT : WAVES_match_list;

  begin
    if WPL.WPL( I ).M_CONTROL = SAMPLE_START then
      RESULT.M_FLAGS( I ) := TRUE;
    elsif WPL.WPL( I ).M_CONTROL = SAMPLE then
      if not
        IS_EQUAL( CODE, Logic_value'val( WPL.WPL( I ).L_VALUE ) ) then
```

```
        RESULT.M_FLAGS( I ) := FALSE;
      end if;
    end if;
    if WPL.DELAY_LOGIC >= 0 then
      RESULT.D_FLAG := IS_EQUAL( CODE,
                        Logic_value'val( WPL.DELAY_LOGIC ));
    end if;
    return RESULT;
end Compare;


--
-- This function compares each Sim_code in the Sim_code_vector to the
-- corresponding elements (given by PIN_SET) of the given WAVES port
-- list and returns a WAVES_match_list indicating the results.
--
function Compare( signal WPL     : WAVES_port_list;
                  VECTOR  : Sim_code_vector;
                  PIN_SET : Pinset := ALL_PINS )
          return WAVES_match_list is

constant ASCENDING : Boolean := VECTOR'left = VECTOR'low;
variable RESULT    : WAVES_match_list;
variable I         : natural := VECTOR'left;

begin
    for PIN in Pinset'range loop
      if PIN_SET( PIN ) then
        RESULT := Compare( WPL, VECTOR( I ), PIN );
        if ASCENDING then
          I := I + 1;
        else
          I := I - 1;
        end if;
      end if;
    end loop;
```

```
    return RESULT;
  end Compare;

  --
  -- This procedure assigns the element (given by PIN) of the WAVES
  -- match list to the values specified in the WAVES_match_list that are
  -- given by PIN.
  --
  procedure Match_Assign( signal WML   : inout WAVES_match_list;
                          VALUE : in   WAVES_match_list;
                          PIN   : in   Test_Pins ) is

  variable I : natural := Test_pins'pos( PIN ) + 1;

  begin
    WML.M_FLAGS( I ) <= VALUE.M_FLAGS( I );
    WML.D_FLAG <= VALUE.D_FLAG;
  end Match_Assign;

  --
  -- This procedure assigns all of the elements (given by PIN_SET) of
  -- the WAVES match list to the values specified in the WAVES_match_list
  -- that are members of PIN_SET.
  --
  procedure Match_Assign( signal WML     : inout WAVES_match_list;
                          VALUE   : in   WAVES_match_list;
                          PIN_SET : in   Pinset := ALL_PINS ) is

  begin
    for PIN in Pinset'range loop
      if PIN_SET( PIN ) then
        Match_Assign( WML, VALUE, PIN );
      end if;
    end loop;
  end Match_Assign;
```

```
--
-- This procedure compares a Sim_code to an element (given by PIN) of
-- the WAVES port list and assigns the corresponding element of the
-- WAVES match list to the result.
--
procedure Compare( signal WPL  : inout WAVES_port_list;
                   signal WML  : inout WAVES_match_list;
                       CODE : in   Sim_code;
                       PIN  : in   Test_pins ) is

begin
   Match_Assign( WML, Compare( WPL, CODE, PIN ), PIN );
end Compare;



--
-- This procedure compares each Sim_code in the Sim_vector to the
-- elements (given by PIN_SET) of the WAVES port list and assigns
-- the corresponding elements of the WAVES match list to the results.
--
procedure Compare( signal WPL     : inout WAVES_port_list;
                   signal WML     : inout WAVES_match_list;
                       VECTOR  : in   Sim_code_vector;
                       PIN_SET : in   Pinset := ALL_PINS ) is

begin
   Match_Assign( WML, Compare( WPL, VECTOR, PIN_SET ), PIN_SET );
end Compare;


--
-- This procedure returns the tag string associated with the given
-- pin from the WAVES port list.
--
procedure Get_Tag( signal WPL : inout WAVES_port_List;
                       TAG :  out String;
                       PIN : in   Test_pins ) is
```

```
  constant INDEX     : natural := Test_pins'pos( PIN ) + 1;

  begin
    wait until WPL.TAG_FLAG;
    if WPL.WPL( INDEX ).TAG_FLAG then
      if TAG'length < WPL.TAG_LEN then
        TAG( 1 to TAG'length ) := WPL.TAG_STR( 1 to TAG'length );
      else
        TAG( 1 to WPL.TAG_LEN ) := WPL.TAG_STR( 1 to WPL.TAG_LEN );
      end if;
    end if;
  end Get_Tag;

  --
  -- This procedure returns the tag string associated with the given
  -- pin set from the WAVES port list.
  --
  procedure Get_Tag( signal WPL     : inout WAVES_port_List;
                     TAG     :  out String;
                     PIN_SET : in    Pinset := ALL_PINS ) is

  variable IS_TAGGED : Boolean := TRUE;

  begin
    wait until WPL.TAG_FLAG;
    for PIN in PIN_SET'range loop
      if PIN_SET( PIN ) /= WPL.WPL( Test_pins'pos( PIN ) + 1 ).TAG_FLAG
      then
        IS_TAGGED := FALSE;
        exit;
      end if;
    end loop;
    if IS_TAGGED then
      if TAG'length < WPL.TAG_LEN then
        TAG( 1 to TAG'length ) := WPL.TAG_STR( 1 to TAG'length );
```

```
        else
          TAG( 1 to WPL.TAG_LEN ) := WPL.TAG_STR( 1 to WPL.TAG_LEN );
        end if;
      end if;
    end Get_Tag;

    --
    -- This procedure returns the tag string associated with the current
    -- WAVES port list.
    --
    procedure Get_Tag( signal WPL : inout WAVES_port_List;
                       TAG :  out String ) is

    begin
      wait until WPL.TAG_FLAG;
        if TAG'length < WPL.TAG_LEN then
          TAG( 1 to TAG'length ) := WPL.TAG_STR( 1 to TAG'length );
        else
          TAG( 1 to WPL.TAG_LEN ) := WPL.TAG_STR( 1 to WPL.TAG_LEN );
        end if;
    end Get_Tag;

end WAVES_VHDL_Interface;
```

Appendix B: Four Bit ALU VHDL Model

The following VHDL model of the four bit ALU is taken from the National Semiconductor databook [3]. This model is presented in three listings, the entity declaration, the architecture body, and the support package that defines types and functions that define the CMOS performance characteristics for various operating conditions.

Each Listing represents a separate source file. Listing B1 lists the source code for the VHDL entity declaration. This entity is defined with two generics, VOLTAGE and OPERATING_TEMPERATURE. The values of these generics are used to select the appropriate timing delays from the performance characteristics table listed in Listing B3.

Listing B1: Entity Declaration

```
------------------------------------------------------------------------
--
-- This design unit defines the signal interface for the 181 ALU.  This
-- device interface provides two simulation generics: Voltage and
-- Temperature.
--
------------------------------------------------------------------------
--
use WORK.STD_LOGIC_1164.all;
use WORK.MM54HC181_CMOS_PERFORMANCE_CHARACTERISTICS.all;
entity mm54hc181 is
   generic( VOLTAGE : Volts := MID;
            OPERATING_TEMPERATURE : Temperature := TYPICAL );
   port( A_INPUTS, B_INPUTS, SELECT_LINES : in Std_ulogic_vector;
         CN, M : in Std_ulogic;
         F_OUTPUTS : out Std_ulogic_vector;
         AEQB, CN_4, NOTG, NOTP : out Std_ulogic );
end mm54hc181;
```

Appendix B

Listing B2 lists the VHDL architecture body for the behavioral model of the four bit ALU. The logic equations for this model are derived from the logic diagram of the National Semiconductor MM54HC181/MM74HC181 Arithmetic Logic Unit/Function Generator description. The Get_Mode function is derived from Table 1 of the general description section of the National Semiconductor MM54HC181/MM74HC181 Arithmetic Logic Unit/Function Generator description.

Listing B2: Architecture Body.

```
-------------------------------------------------------------------------
--
-- This design unit defines a behavioral model for the 181 ALU.
--
-------------------------------------------------------------------------
--
use WORK.MM54HC181_CMOS_PERFORMANCE_CHARACTERISTICS.all;
architecture behavioral of mm54hc181 is

  --
  -- This function determines the chip mode based on the Mode
  -- bit and the select bits.  Valid modes are SUM, DIF, and LOGIC.
  --
  function Get_Mode( M, S0, S1, S2, S3 : in Std_ulogic ) return Mode is

  begin
    if M = '1' then
      return LOGIC;
    elsif ( S3 = '0' and S2 = '0' and S1 = '1' and S0 = '1' ) or
          ( S3 = '0' and S2 = '1' and S1 = '1' and S0 = '0' ) or
          ( S3 = '0' and S2 = '1' and S1 = '1' and S0 = '1' ) or
          ( S3 = '1' and S2 = '0' and S1 = '1' and S0 = '1' ) or
          ( S3 = '1' and S2 = '1' and S1 = '1' and S0 = '1' ) then
      return DIF;
    else
```

```
        return SUM;
      end if;
    end Get_Mode;

    --
    -- Declare internal signals.
    --
    signal N0, N1, N2, N3, N4, N5, N6, N7, N8, N9,
        N10, N11, N12, N13, N14, N15, N16, N17 : Std_ulogic;

    signal CHIP_MODE : Mode;

  begin
    --
    CHIP_MODE <= Get_Mode( M, SELECT_LINES( 0 ), SELECT_LINES( 1 ),
                    SELECT_LINES( 2 ), SELECT_LINES( 3 ) );
    --
    -- Assign input nodes.
    --
    N0  <= ( A_INPUTS( 3 ) and B_INPUTS( 3 ) and SELECT_LINES( 3 ) ) nor
        ( A_INPUTS( 3 ) and not B_INPUTS( 3 ) and SELECT_LINES( 2 ) );
    N1  <= not(( not B_INPUTS( 3 ) and SELECT_LINES( 1 ) ) or
        ( B_INPUTS( 3 ) and SELECT_LINES( 0 ) ) or ( A_INPUTS( 3 ) ));
    N2  <= ( A_INPUTS( 2 ) and B_INPUTS( 2 ) and SELECT_LINES( 3 ) ) nor
        ( A_INPUTS( 2 ) and not B_INPUTS( 2 ) and SELECT_LINES( 2 ) );
    N3  <= not(( not B_INPUTS( 2 ) and SELECT_LINES( 1 ) ) or
        ( B_INPUTS( 2 ) and SELECT_LINES( 0 ) ) or ( A_INPUTS( 2 ) ));
    N4  <= ( A_INPUTS( 1 ) and B_INPUTS( 1 ) and SELECT_LINES( 3 ) ) nor
        ( A_INPUTS( 1 ) and not B_INPUTS( 1 ) and SELECT_LINES( 2 ) );
    N5  <= not(( not B_INPUTS( 1 ) and SELECT_LINES( 1 ) ) or
        ( B_INPUTS( 1 ) and SELECT_LINES( 0 ) ) or ( A_INPUTS( 1 ) ));
    N6  <= ( A_INPUTS( 0 ) and B_INPUTS( 0 ) and SELECT_LINES( 3 ) ) nor
        ( A_INPUTS( 0 ) and not B_INPUTS( 0 ) and SELECT_LINES( 2 ) );
    N7  <= not(( not B_INPUTS( 0 ) and SELECT_LINES( 1 ) ) or
        ( B_INPUTS( 0 ) and SELECT_LINES( 0 ) ) or ( A_INPUTS( 0 ) ));
    --
```

```
-- Assign internal nodes.
--
N8  <= not( N0 and N2 and N4 and N6 and CN );
N9  <= N0 xor N1;
N10 <= not(( N2 and N4 and N6 and CN and not M) or
        ( N2 and N4 and N7 and not M ) or
        ( N2 and N5 and not M ) or
        ( N3 and not M ));
N11 <= N2 xor N3;
N12 <= not(( N4 and N6 and CN and not M ) or
        ( N4 and N7 and not M ) or
        ( N5 and not M ));
N13 <= N4 xor N5;
N14 <= ( N6 and CN and not M ) nor ( N7 and not M );
N15 <= N6 xor N7;
N16 <= not( CN and not M );
--
-- Assign output signals.
--
N17   <= not( N1 or ( N0 and N3 ) or ( N0 and N2 and N5 ) or
       ( N0 and N2 and N4 and N7 ));

NOTG  <= N17 after DELAY_TABLE( CHIP_MODE, NOT_G, VOLTAGE,
                    OPERATING_TEMPERATURE );

CN_4  <= not N17 or not N8 after
        DELAY_TABLE( CHIP_MODE, C_N_4, VOLTAGE,
                OPERATING_TEMPERATURE );

NOTP  <= not( N0 and N2 and N4 and N6 ) after
        DELAY_TABLE( CHIP_MODE, NOT_P, VOLTAGE,
                OPERATING_TEMPERATURE );

F_OUTPUTS( 0 ) <= N15 xor N16 after
        DELAY_TABLE( CHIP_MODE, NOT_F, VOLTAGE,
                OPERATING_TEMPERATURE );
```

```
    F_OUTPUTS( 1 ) <= N13 xor N14 after
            DELAY_TABLE( CHIP_MODE, NOT_F, VOLTAGE,
                    OPERATING_TEMPERATURE );

    F_OUTPUTS( 2 ) <= N11 xor N12 after
            DELAY_TABLE( CHIP_MODE, NOT_F, VOLTAGE,
                    OPERATING_TEMPERATURE );

    F_OUTPUTS( 3 ) <= N9 xor N10 after
            DELAY_TABLE( CHIP_MODE, NOT_F, VOLTAGE,
                    OPERATING_TEMPERATURE );

    AEQB  <= ( N9 xor N10 ) and ( N11 xor N12 ) and
          ( N13 xor N14 ) and ( N15 xor N16 ) after
            DELAY_TABLE( CHIP_MODE, A_EQ_B, VOLTAGE,
                    OPERATING_TEMPERATURE );

end behavioral;
```

Listing B3 lists the VHDL support package MM54HC181_CMOS_PERFORMANCE_CHARACTERISTICS. This package declares types for Volts, Temperature, Mode, and Signal_name, and defines a look-up table that defines the CMOS timing characteristics for the four bit ALU. The data contained in this table is taken from the AC Electrical Characteristics table from National Semiconductor MM54HC181/MM74HC181 Arithmetic Logic Unit/Function Generator description[4].

Listing B3: Supporting Package.

```
------------------------------------------------------------------------
-- Package : MM54HC181_CMOS_PERFORMANCE_CHARACTERISTICS
--
-- Description:
--
-- This package exports types and constants required to represent the
-- performance characteristics for the CMOS 181 ALU.  The data in this
-- package is from the National Semiconductor MM54HC181 ALU specification.
--
------------------------------------------------------------------------
--
package mm54hc181_cmos_performance_characteristics is

   type Volts is ( LOW, MID, HIGH );
   --          2.0, 4.5, 6.0

   type Temperature is ( TYPICAL, G_LOW, G_MID, G_HIGH );
   --                    25C,   25C,   85C,   125C

   type Mode is ( SUM, DIF, LOGIC );

   type Signal_Name is ( A_EQ_B, C_N_4, NOT_F, NOT_G, NOT_P );

   type Delay_matrix is array(
         Mode'left to Mode'right,
          Signal_name'left to Signal_name'right,
         Volts'left to Volts'right,
         Temperature'left to Temperature'right ) of Time;

   constant DELAY_TABLE : Delay_matrix :=
     ( -- SUM
         ( -- AEQB
           ( ( 0 ns, 0 ns, 0 ns, 0 ns ),
```

```
        ( 0 ns, 0 ns, 0 ns, 0 ns ),
        ( 0 ns, 0 ns, 0 ns, 0 ns ) ),
      -- CN_4
      ( ( 110 ns, 250 ns, 325 ns, 375 ns ),
        ( 35 ns, 50 ns, 63 ns, 75 ns ),
        ( 30 ns, 43 ns, 53 ns, 65 ns ) ),
      -- NOTF
      ( ( 115 ns, 240 ns, 300 ns, 360 ns ),
        ( 35 ns, 48 ns, 60 ns, 72 ns ),
        ( 30 ns, 41 ns, 51 ns, 61 ns ) ),
      -- NOTG
      ( ( 55 ns, 120 ns, 160 ns, 200 ns ),
        ( 17 ns, 24 ns, 30 ns, 36 ns ),
        ( 14 ns, 20 ns, 25 ns, 30 ns ) ),
      -- NOTP
      ( ( 70 ns, 150 ns, 189 ns, 224 ns ),
        ( 20 ns, 30 ns, 38 ns, 45 ns ),
        ( 17 ns, 26 ns, 32 ns, 38 ns ) ) ),

  -- DIF
    ( -- AEQB
      ( ( 120 ns, 280 ns, 350 ns, 420 ns ),
        ( 40 ns, 56 ns, 70 ns, 84 ns ),
        ( 35 ns, 48 ns, 60 ns, 72 ns ) ),
      -- CN_4
      ( ( 120 ns, 280 ns, 350 ns, 420 ns ),
        ( 40 ns, 56 ns, 70 ns, 84 ns ),
        ( 35 ns, 48 ns, 60 ns, 72 ns ) ),
      -- NOTF
      ( ( 120 ns, 275 ns, 344 ns, 344 ns ),
        ( 40 ns, 55 ns, 69 ns, 83 ns ),
        ( 34 ns, 47 ns, 59 ns, 69 ns ) ),
      -- NOTG
      ( ( 70 ns, 150 ns, 189 ns, 224 ns ),
        ( 20 ns, 30 ns, 38 ns, 45 ns ),
        ( 17 ns, 26 ns, 32 ns, 38 ns ) ),
```

```
            -- NOTP
            ( ( 70 ns, 150 ns, 189 ns, 224 ns ),
              ( 20 ns, 30 ns, 38 ns, 45 ns ),
              ( 17 ns, 26 ns, 32 ns, 38 ns ) ) ),

          -- LOGIC
            ( -- AEQB
              ( ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ) ),
              -- CN_4
              ( ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ) ),
              -- NOTF
              ( ( 120 ns, 275 ns, 344 ns, 344 ns ),
                ( 40 ns, 55 ns, 60 ns, 83 ns ),
                ( 34 ns, 47 ns, 59 ns, 69 ns ) ),
              -- NOTG
              ( ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ) ),
              -- NOTP
              ( ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ),
                ( 0 ns, 0 ns, 0 ns, 0 ns ) ) ) );


end mm54hc181_cmos_performance_characteristics;
```

Appendix C: The WAVES Dataset

This Appendix lists all of the WAVES source code that describes the test vectors for the MM54HC181 four bit ALU. These vectors are taken from MIL-M-38510/11C 19 May 1978 Table III[4]. Each Listing in this Appendix represents a separate source file. Listing C1 lists the WAVES header file. This file serves to identify the WAVES dataset and specify its construction.

Listing C1: The WAVES Header File.

```
-------------------------------------------------------------------------
--
-- This is the header file for the WAVES dataset used to provide the
-- stimulus/response data for simulating the 181 ALU.
--
-------------------------------------------------------------------------
--
-- Dataset identification information.
--
TITLE              181 ALU Vector Generator
DEVICE_ID          181 ALU
DATE               21-AUG-1992
ORIGIN             MIL-M-38510/11C 19 May 1978 Table III
AUTHOR             Rome Laboratory ERDD
AUTHOR             James P. Hanna
DATE               06-DEC-1993
ORIGIN             Removal of cyclic elaboration problem
AUTHOR             Rome Laboratory ERDD
AUTHOR             James P. Hanna
OTHER              Rearranged code to remove cyclic references.
--
-- Dataset construction information.
--
WAVES_FILENAME     mm54hc181_waves_logic_.vhd          WORK
```

```
WAVES_FILENAME     mm54hc181_waves_logic.vhd                WORK
WAVES_FILENAME     mm54hc181_waves_codes_.vhd               WORK
WAVES_FILENAME     mm54hc181_waves_dut_.vhd                 WORK
WAVES_UNIT         WAVES_INTERFACE                          WORK
WAVES_UNIT         WAVES_OBJECTS                            WORK
WAVES_FILENAME     mm54hc181_waveform_generator_.vhd        WORK
WAVES_FILENAME     mm54hc181_waveform_generator.vhd         WORK
--
EXTERNAL_FILENAME  mm54hc181_waves_patterns.txt             VECTORS
--
WAVEFORM_GENERATOR_PROCEDURE
WORK.WAVEFORM_GENERATOR.WAVEFORM
```

Listing C2 lists the code that declares the required, WAVES user defined type Test_pins. This type enumerates the names of all of the functional pins of the device interface.

Listing C2: The Test_Pins Declaration.

```
package MM54HC181_WAVES_DUT is

  type Test_pins is ( S3, S2, S1, S0, CN, M,
                      NOTF0, NOTF1, NOTF2, NOTF3, AEQB, NOTP, CN_4,
                      NOTG, B3, A3, B2, A2, B1, A1, B0, A0 );

end MM54HC181_WAVES_DUT;
```

Listing C3 lists the code that declares the required, WAVES user defined constant PIN_CODES. This constant defines the codes that will be used in the external file to identify which frame to apply to each pin in Test_pins for each vector.

Listing C3: The Pin_Codes Declaration.

```
package MM54HC181_WAVES_Codes is

   constant PIN_CODES : String := "UXBALH";

end MM54HC181_WAVES_Codes;
```

Listing C4 lists the code that declares the required, WAVES user defined type Logic_value. This type defines names for each type of event that can occur on the waveform. This package also declares the required, user defined function Value_Dictionary.

Listing C4: The Logic_value Declaration.

```
library WAVES_STD;
use WAVES_STD.WAVES_STANDARD.all;
package MM54HC181_WAVES_Logic is

   type Logic_value is
        (LV_UNKNOWN, DONT_CARE,
         DRIVE_0, DRIVE_1,
          SENSE_0, SENSE_1);

   function Value_Dictionary( VALUE : in Logic_value )
            return Event_value;

end MM54HC181_WAVES_Logic;
```

Appendix C

Listing C5 lists the code that defines the required, WAVES user defined function Value_Dictionary. This function defines the semantics of each of the logic values that are declared in Listing C4.

Listing C5: The Value_Dictionary Declaration.

```
package body MM54HC181_WAVES_Logic is

  function Value_Dictionary( VALUE : in Logic_value )
                return Event_value is

  begin
    case VALUE is
    when LV_UNKNOWN =>
      return state = UNKNOWN and
             strength = UNKNOWN and
             direction = UNKNOWN and
             relevance = UNKNOWN;
    when DONT_CARE =>
      return UNSPECIFIED;
    when DRIVE_0 =>
      return state = LOW and
             strength = DRIVE and
             direction = STIMULUS;
    when DRIVE_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;

end MM54HC181_WAVES_Logic;
```

Listing C6 lists the package specification in which the required waveform generator procedure is declared. This procedure provides the interface to the WAVES port list. Additionally, this package defines all of the pin sets of interest.

Listing C6: The Waveform_Generator Package Specification.

```
use WORK.MM54HC181_WAVES_Logic.all;
use WORK.MM54HC181_WAVES_Codes.all;
use WORK.MM54HC181_WAVES_DUT.all;
use WORK.WAVES_INTERFACE.all;
use WORK.WAVES_OBJECTS.all;
package Waveform_Generator is

   procedure Waveform( signal CONNECT : inout WAVES_port_list );

   constant A_PINS : Pinset := New_Pinset( ( A0, A1, A2, A3 ) );

   constant B_PINS : Pinset := New_Pinset( ( B0, B1, B2, B3 ) );

   constant S_PINS : Pinset := New_Pinset( ( S0, S1, S2, S3 ) );

   constant F_PINS : Pinset := New_Pinset( ( NOTF0, NOTF1, NOTF2, NOTF3 ) );

end Waveform_Generator;
```

Appendix C

Listing C7 lists the package body in which the waveform generator procedure and frame set function implementation are defined. The frame set function defines the mapping between the pin codes declared in Listing C3 and (sets) of logic values (events) and the time of their occurrence within a given slice. The waveform generator procedure reads the external file and the Apply procedure constructs the waveform one slice at a time until the entire external file has been read. The sequence of events that are scheduled on each pin of the CONNECT parameter are defined by the mappings described in the frame set function.

Listing C7: The Waveform_Generator Package Body.

```
use STD.TEXTIO.all;
package body Waveform_Generator is


  function MM54HC181_Frame_Set( STRB_START, STRB_STOP : Time )
    return Frame_Set is

  begin
   return
    New_Frame_Set( 'U', Frame_Event( ( LV_UNKNOWN, Etime( 0 ns ) ) ) ) +
    New_Frame_Set( 'X', Frame_Event( ( DONT_CARE, Etime( 0 ns ) ) ) ) +
    New_Frame_Set( 'B', Frame_Event( ( DRIVE_0, Etime( 0 ns ) ) ) ) +
    New_Frame_Set( 'A', Frame_Event( ( DRIVE_1, Etime( 0 ns ) ) ) ) +
    New_Frame_Set( 'L', Frame_Elist( (( DONT_CARE, Etime( 0 ns ) ),
                                     ( SENSE_0, Etime( STRB_START ) ),
                                     ( DONT_CARE, Etime( STRB_STOP ))))) +
    New_Frame_Set( 'H', Frame_Elist( (( DONT_CARE, Etime( 0 ns ) ),
                                     ( SENSE_1, Etime( STRB_START ) ),
                                     ( DONT_CARE, Etime( STRB_STOP ) ))));
  end MM54HC181_Frame_Set;


  procedure Waveform( signal  CONNECT   : inout WAVES_port_list ) is
```

```
    file PATTERNS : TEXT is in "VECTORS";

    variable VECTOR : File_slice := New_File_Slice;

    constant STRB_START : Time := 250 ns;

    constant STRB_STOP : Time := 300 ns;

    constant MM54HC181_FRAME_SET_ARRAY : Frame_set_array :=
            New_Frame_Set_Array(
              MM54HC181_Frame_Set( STRB_START, STRB_STOP ), ALL_PINS );

    variable TIMING_DATA : Time_data :=
            New_Time_Data( MM54HC181_FRAME_SET_ARRAY );

    begin

      loop
        Read_File_Slice( PATTERNS, VECTOR );
        exit when VECTOR.END_OF_FILE;
        Apply( CONNECT, VECTOR.CODES.all,
                Delay( VECTOR.FS_TIME ), TIMING_DATA );
      end loop;

    end Waveform;
end Waveform_Generator;
```

Listing C8 lists the WAVES external file. This file contains the sequences of pin codes that comprise the vector set to be read and applied by the WAVES waveform generator procedure. Each vector, including the timing information, represents one slice of the waveform. The timing information at the end of each pattern indicates the duration of each slice.

Listing C8: The WAVES External File.

```
%
% External File: MM54HC181_WAVES_PATTERNS.TXT
%
BBBBBBHLLLLLHLBBBBBBBB : 500 ns;
BABAAAHHHHHLHLBBBBBBBB : 500 ns;
AABBAAHHHHHHLHBABBBBBB : 500 ns;
ABBABBHLLLLHLHABABBAAA : 500 ns;
BABABBHHLLLHLHABBABBBA : 500 ns;
AAAABBLLLLLLLHABABABBB : 500 ns;
AABAABHHLLLLHLBBBBABAB : 500 ns;
ABBBAAHHHLLLHLBABBBBBB : 500 ns;
AABBABLLHHLHHLBBBABABB : 500 ns;
AAABBBHHHHHHHHLBBABBABA : 500 ns;
BABBAAHHHLLLHLAABBBBBB : 500 ns;
ABBABBHHLHLLHLABBBABBB : 500 ns;
AABAABLLHLLHHLBBBBABBA : 500 ns;
AABBABLLHLLHHLABBBBABB : 500 ns;
BABABBHHHHHHHLABABBBBA : 500 ns;
BABAABLHHHLLHLAAAAAABB : 500 ns;
AABAABLLLHLHHLBBABABBA : 500 ns;
BAABBBHHHHHHHLBBBBABBA : 500 ns;
AAAABBHLLLLHLHABABBBBA : 500 ns;
BABBABLLLHLHHLBBBABBBB : 500 ns;
AABAABLLLHLHHLBBABAABB : 500 ns;
ABBAABLLHHLLHLABBABBBB : 500 ns;
AAAAABLLLLLHLHABABABBA : 500 ns;
ABBAABHHHHHLHHABABABAB : 500 ns;
AABAABLLLLLHLHABAABBBB : 500 ns;
BABBABHHHHHLHHAAAAAAAA : 500 ns;
ABBBAAHHHHHHLHAABBBBBB : 500 ns;
BBBAABHHHHHLHHABABABBA : 500 ns;
ABBBBAHHHHHHLHAAAABBAB : 500 ns;
ABAAAALHLHLHLHAAABAABA : 500 ns;
```

AAABBAHLLHLHHLABBBBBBA : 500 ns;
AAABABLLLLLHLHBBBBAAAB : 500 ns;
BBBBBBHHHLLLHLBBBABABB : 500 ns;
AAAABAHLHHLHLHBABABBBA : 500 ns;

References

[1] "IEEE Standard for Waveform and Vector Exchange (WAVES)," Institute of Electrical and Electronics Engineers, IEEE Std 1029.1-1991.

[2] "IEEE Standard VHDL Language Reference Manual, "Institute of Electrical and Electronics Engineers, IEEE Std 1076-1987.

[3] "MM54HC/74HC High Speed microCMOS Logic Family Databook," National Semiconductor Corporation, 1883.

[4] "Military Specification, Microcircuits, Digital, TTL Arithmetic Logic Units/Function Generators, Monolithic Silicon," MIL-M-38510/11C, 19 May 1978.

Bibliography

Robert G. Hillman, "WAVES: A Simulation and Tester View," Proc. IEEE International Test Conference, 1985.

"Prototype WAVES Standard VHDL Packages," provided to the WAVES Analysis and Standardization Group, of the IEEE Standards Coordinating Committee 20 and the IEEE Design Automation Standards Subcommittee, version 3.4.1 1 June 1991.

*U.S. GOVERNMENT PRINTING OFFICE: 1994-510-117-50004

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.