Chapter 7

7.0 WAVES Testbenches

7.1 Introduction

This chapter will address the generation of testbenches in accordance with the Waveform and Vector Exchange Specification (WAVES IEEE-STD-1029.1). This chapter is not designed to teach an engineer everything there is to know about WAVES. Rather, it is intended to provide an introduction to WAVES, giving sufficient information to allow a design engineer to begin writing WAVES testbenches.

7.2 WAVES Standard Packages

The WAVES standard identifies one utility package and three standard packages as follows. The declarations for these packages is included in IEEE-STD-1029.1. A listing of data type declarations and functions is provided in the following paragraphs. It should be noted that some of the character formatting in these files (i.e. upper versus lower case letters) differs from the format provided in the original files. This is to maintain consistency with the formatting conventions of this document. Since VHDL identifiers/reserved words are case insensitive, this does not affect the meaning of the code.

Package Name	Package Type
WAVES_SYSTEM	Utility
WAVES_STANDARD	Standard
WAVES_INTERFACE	Standard
WAVES_OBJECTS	Standard

7.2.1 The WAVES_SYSTEM Package

The WAVES_SYSTEM package defines system dependent information and private types used by the WAVES packages. Table 7.2.1-1 identifies the data types which are declared in this package. Refer to IEEE-STD-1029.1 for a listing of the WAVES_SYSTEM package declaration. It should be noted that the following declarations are not included in the reserved word listing for WAVES. This is to provide consistency with the listing provided in IEEE-STD-1029.1, Appendix A. Because of this, these words will not appear in upper case bold face print in the code listings which follow.

7.2.2 The WAVES_STANDARD Package

The WAVES_STANDARD package provides type definitions and functions used to describe EVENT_VALUE objects as defined in the WAVES_SYSTEM package. The EVENT_VALUE type is system dependent. IEEE-STD-1029.1 contains a listing of the WAVES_STANDARD package declaration. Table 7.2.2-1 contains a listing of data elements defined in this package.

7.2.3 The WAVES_INTERFACE Package

The WAVES_INTERFACE package requires visibility to the user defined LOGIC_VALUE data type. This data type will be discussed in later sections. Once the user has selected a LOGIC_VALUE data type with which they are comfortable, the WAVES_INTERFACE package can be analyzed for all designs which utilize the specified LOGIC_VALUE. IEEE-STD-1029.1 contains the package declaration for the WAVES_INTERFACE package. Table 7.2.3-1 provides a listing of the elements defined in this declaration. The user should note the requirement for adding visibility to the LOGIC_VALUE data type in this package.

7.2.3.1 The WAVES_INTERFACE package contains two private constant declarations and a private function which are used by functions within the package body. These items are identified as follows.

Declarations	Used in
CONSTANT CODE_LEN	function **NEW_FRAME_SET**
	function "+" (2)
CONSTANT LOGIC_LEN	function **NEW_LOGIC_MAP**(1)
	function "+" (1)
FUNCTION COPY_EVENT	function **FRAME_ELIST**(1)

Declaration	Data Type
state_type	enumerated type
state_value	record type
strength_type	enumerated type
strength_value	record type
direction_type	enumerated type
relevance_type	enumerated type
unknown_keyword	enumerated type
state_keyword	enumerated type
strength_keyword	enumerated type
direction_keyword	enumerated type
relevance_keyword	enumerated type
system_direction_summary	enumerated type
operator_type	enumerated type
operator_kind	enumerated type
event_value_element	record type
system_event_value	event_value_element array type
system_delay_time_type	enumerated type
system_delay_time_basis	record type
system_event_time	record type
system_event	record type
system_event_time_basis	record type
system_tagged_time	enumerated type
system_tagged_flag	enumerated type
system_tagged_temp	record type
system_tagged_event	record type
null_temp	constant
null_event	constant
system_frame	system_tagged_event array
system_frame_list	system_tagged_event array
system_frame_set	system_tagged_event array
system_logic_map	system_tagged_event array
system_frame_set_array	system_tagged_event array
system_match_control	enumerated type
system_waves_port	record type
system_tag_string	string subtype
system_waves_port_list	system_waves_port array type
system_waves_match_list	boolean arraytype

Declaration	Subprogram Type
promote	function (4)

Table 7.2.1-1. WAVES_SYSTEM Package Declarations.

7.2.3.2 The constant CODE_LEN is dependent upon the length of the PIN_CODES string. Therefore, the PIN_CODES string must be made visible to this package in addition to LOGIC_VALUE.

7.2.4 The WAVES_OBJECTS Package

Declaration	Data Type
EVENT_VALUE	waves_system.system_event_value subtype
DIRECTION_SUMMARY	waves_system.system_direction_summary subtype
DIRECTION_LIST	direction_summary array type

Declaration	Subprogram Type
UNSPECIFIED	function (2)
UNKNOWN	function (2)
STATE	function
LOW	function (2)
MIDBAND	function (2)
HIGH	function (2)
STRENGTH	function
DISCONNECTED	function (2)
CAPACITIVE	function (2)
RESISTIVE	function (2)
DRIVE	function (2)
SUPPLY	function (2)
DIRECTION	function
STIMULUS	function (2)
RESPONSE	function (2)
RELEVANCE	function
REQUIRED	function
PREDICTED	function
OBSERVED	function
COMPOUND	function
">"	function (2)
"<"	function (2)
"="	function (8)
"/="	function (2)
">="	function (2)
"<="	function (2)
"not"	function
"and"	function
"or"	function

Table 7.2.2-1. WAVES_STANDARD Package Declarations.

The source code for the WAVES_OBJECTS package is contained in IEEE-STD-1029.1. The user should note that this package requires visibility to both LOGIC_VALUE, TEST_PINS and PIN_CODES data types. Because the TEST_PINS data type is unique for each design, this package will have to be unique for each application. Table 7.2.4-1 containsw the data type and function declarations made in this package.

7.3 The WAVES Header File.

The WAVES header file is used to specify information which is not available in the other WAVES dataset files, but is required or useful to describe the WAVES dataset completely. This file specifies the names of the WAVES files and references the standard WAVES packages in the proper order for analysis. It also allows for a mapping of logical file names to physical file names for external files. Figure 7.3-1 contains an example WAVES header file for the DQD module example of Chapter 5. The WAVES header file is provided as reference information. It is not used by VHDL development tools to assist with the analysis of a WAVES dataset.

Declaration	Data Type
LOGIC_SET	boolean array type
LOGIC_LIST	logic_value array type
EVENT_TIME_BASIS	waves_system.system_event_time_basis subtype
EVENT_TIME	record type
EVENT	record type
EVENT_LIST	event array type
FRAME	waves_system.system_frame subtype
FRAME_LIST	waves_system.system_frame_list subtype
FRAME_SET	waves_system.system_frame_set subtype
LOGIC_MAP	waves_system.system_logic_map subtype
FRAME_SET_ARRAY	waves_system.system_frame_set_array subtype
DELAY_TIME_BASIS	waves_system.system_delay_time_basis subtype
DELAY_TIME	record type

Declaration	Subprogram Type
NEW_LOGIC_SET	function (3)
ETIME	function (3)
MERGE_ETIME	function
FRAME_EVENT	function (2)
FRAME_ELIST	function (2)
NEW_FRAME_SET	function
NEW_LOGIC_MAP	function (2)
DELAY	function (3)
"+"	function (12)
"-"	function (4)
"*"	function (4)
"/"	function (4)

Table 7.2.3-1. WAVES_INTERFACE Package Declarations.

7.4 The WAVES Logic Value Package

The logic value file contains the declarations and functions of Table 7.4-1. An example file is shown in Figure 7.4-1. This package was used for the FBG and DQD modules on the TIREP project. The primary purpose of this package is to establish the VALUE_DICTIONARY This function is used to establish signal characteristics for the testbench.

7.4.1 Background

Before trying to address the elements of this package, some background on signal characteristics within WAVES, is in order. There are four parameters which establish a signals characteristics. These are as follows. These parameters are declared in the WAVES_SYSTEM package, through the EVENT_VALUE subtype. These parameters will be used in the development of the VALUE_DICTIONARY which will be discussed in the following sections.

Parameter	Declared Values
STATE	LOW, MIDBAND, HIGH
STRENGTH	DISCONNECTED, CAPACITIVE, RESISTIVE, DRIVE, SUPPLY
DIRECTION	STIMULUS, RESPONSE
RELEVANCE	OBSERVED, PREDICTED, REQUIRED

Declaration	Data Type
TIME_DATA	frame_set_array access type
WAVE_TIMING	record type
TIME_DATA_LIST	time_data array type
WAVE_TIMING_LIST	wave_timing array type
TEST_PINS_LIST	test_pins array type
PIN_CODE_STRING	test_pins subtype
PIN_CODE_LIST	pin_code_string array type
INDEX_SLICE	record type
INDEX_SLICE_LIST	index_slice array type
TIMED_SLICE	record type
TIMED_SLICE_LIST	timed_slice array type
WTIME_SLICE	record type
WTIME_SLICE_LIST	wtime_slice array type
PINSET	boolean array type
ALL_PINS	pinset constant type
NO_PINS	pinset constant type
WAVES_PORT_LIST	record type
WAVES_MATCH_LIST	record type
MATCH_CONTROL_TYPE	enumerated type
FILE_SLICE	record type
FILE_SLICE_LIST	file_slice array type

Declaration	Subprogram Type
ETIME	function (3)
DELAY	function (6)
NEW_PINSET	function (3)
NEW_FRAME_SET_ARRAY	function (4)
NEW_TIME_DATA	function
MERGE_STRING	function (6)
APPLY	procedure (8)
TAG	procedure (3)
MATCH	procedure (2)
MATCH	function (2)
NEW_FILE_SLICE	function (2)
READ_FILE_SLICE	procedure (3)
"+"	function

Table 7.2.4-1. WAVES_OBJECTS Package Declarations.

7.4.1.1.1 The values for the STATE parameter are consistent with the naming convention selected. A LOW value will generally be applied to a logic 0 state, a HIGH value would correspondingly be applied to a logic 1 state and MIDBAND could be applied to high impedance or unknown states.

7.4.1.2 The STRENGTH values are ordered with DISCONNECTED being the weakest and SUPPLY being the strongest. The physical meaning of the STRENGTH values is not defined within WAVES. A typical interpretation might define DISCONNECTED as an open circuit, CAPACITIVE, RESISTIVE and DRIVE as input/output signal characteristics and SUPPLY as a power supply source.

7.4.1.3 The DIRECTION parameter has only two possible values, STIMULUS (input) and RESPONSE (output).

```
--************************************************************
--  (c) Copyright 1994 by the
--        Naval Air Warfare Center, Aircraft Division, Indianapolis
--  Source
--        Author(s):            Charles K. Rogers
--        Organization:           NAWC-ADI
--                              Code 306, MS-42
--                              6000 E 21st St
--                              Indianapolis,  IN  46219-2189
--                              Phone:   317-353-3579
--                              EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--        Reference:            MIL-M-28787/212
--        Project:              SHARP   TIREP
--  DESC Certification
--        Status:               TBD
--************************************************************
--  Revision History
--        Version:              1.0
--            Date:             12 May 1994
--            Comments:         Original Release
--************************************************************
--  Module Description
--        File:                 dqd_hdr.wav
--        Module  Name(s):      none
--        Constraints:          none
--        Limitations:          none
--        I/O  Format(s):       none
--        Purpose and Use:      This VHDL module describes the contents of the
--            WAVES dataset for the FBG standard hardware module. This
--            dataset is based upon Table XIX in the M28787/212 specification.
--        Notes:                none
--************************************************************
--  Standard Libraries/Packages
--        none
--  Associated Packages (order of analysis implied)
--        none
--  Component  Models
--        none
--  Platform:                   486/33MHz PC
--  Software/Version:           V-System for Windows, Version 3.0
--************************************************************
--  Design Specification Elements
--        all                   information  only
--************************************************************
title                           DQD  module  WAVES  dataset
corporate_author                NAWC-ADI
individual_author               Charles  K.  Rogers
release_date_and_time           12 May 1994
origin                          MIL-M-28787/212
```

Figure 7.3-1. A WAVES Header File for the DQD Module.

device_id	DQD SEM module	
-- *Waves files*		
waves_filename	wav_logp.vhd	work
waves_unit	**WAVES_INTERFACE**	work
waves_filename	wav_frmp.vhd	work
waves_filename	dqd_dutp.vhd	work
waves_unit	**WAVES_OBJECTS**	work
waves_filename	dqd_wavp.vhd	work
-- *Waveform external data file*		
external_filename	dqd_vec.dat	data
-- *Waveform generation procedure*		
waveform_generator_procedure		work.dqd_wav.dqdwav

Figure 7.3-1 continued. A WAVES Header File.

Declaration	Data Type
LOGIC_VALUE	enumerated type
PIN_CODES	constant string type
Declaration	Subprogram Type
VALUE_DICTIONARY	function

Table 7.4-1. WAVES Logic Value Package Declarations.

7.4.1.4 The values of RELEVANCE are defined as follows. A value of OBSERVED is one that is detected during simulation and/or during testing of the physical unit or device. An OBSERVED value is not required. A value of PREDICTED identifies an event as being calculated or expected, but not required. A value of REQUIRED identifies a value which is required for the unit or device to meet it’s specification.

7.4.2 The LOGIC_VALUE Data Type

The LOGIC_VALUE data type is a user defined enumerated data type which lists the logic values that will be defined in the VALUE_DICTIONARY. Following is the LOGIC_VALUE declaration provided in the "wav_logic" example package below.

TYPELOGIC_VALUEIS (ui,ud,lo,hi,hz,wl,wh,lo_in,hi_in,hz_in,wl_in,wh_in,
lo_out,hi_out,hz_out,wl_out,wh_out);

7.4.2.1 Each element of the LOGIC_VALUE declaration should conform to legal VHDL naming conventions, however care should be taken to insure that reserved words are not used.

7.4.2.2 This declaration deals with seven basic states. They are uninitialized (ui), undefined (ud), logic 0 (lo), logic 1 (hi), high impedance (hz), weak low (wl), or weak high (wh). As indicated, the direction of these states may be in, out or indeterminate.

7.4.3 The VALUE_DICTIONARY Function

With the LOGIC_VALUE data type defined, it is now possible to establish the VALUE_DICTIONARY. This function is used to assign signal characteristics to each element declared in LOGIC_VALUE.

7.4.3.1 In the example package, the VALUE_DICTIONARY function first establishes a set of constants ("x_lo", "x_hi", and "x_hz"). Each of these constants is assigned a STATE value and a STRENGTH value.

```
--****************************************************************
--   (c) Copyright 1994 by the
--        Naval Air Warfare Center, Aircraft Division, Indianapolis
--   Source
--        Author(s):             Charles K. Rogers
--        Organization:            NAWC-ADI
--                               Code 306, MS-42
--                               6000 E 21st St
--                               Indianapolis,  IN  46219-2189
--                               Phone:   317-353-3579
--                               EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--        Reference:             MIL-M-28787/212
--        Project:               SHARP   TIREP
--   DESC Certification
--        Status:                TBD
--****************************************************************
--   Revision History
--        Version:               1.0
--             Date:             12 May 1994
--             Comments:         Original Release
--****************************************************************
--   Module Description
--        File:                  wav_logp.wav
--        Module  Name(s):       wav_logicpackage
--        Constraints:           none
--        Limitations:           none
--        I/O  Format(s):        none
--        Purpose and Use:       This VHDL package provides the code to event mapping
--             function for the pin codes used in this dataset.
--        Notes:                 none
--****************************************************************
--   Standard Libraries/Packages
--        Waves_Standard         waves standard package
--   Associated Packages (order of analysis implied)
--        none
--   Component  Models
--        none
--   Platform:                   486/33MHz PC
--   Software/Version:           V-System for Windows, Version 3.0
--****************************************************************
--   Design Specification Elements
--        LOGIC_VALUE                 enumerated type
--        VALUE_DICTIONARY       function
--****************************************************************
LIBRARY WAVES_STD;
USE WAVES_STD.WAVES_STANDARD.ALL;
PACKAGE wav_logic IS
--****************************************************************
-- The LOGIC_VALUE used in this package is reasonably encompassing.
```

Figure 7.4-1. A WAVES Logic Value Package.

```
-- In general, this declaration will suffice for most logic designs.
--***************************************************************
    TYPE LOGIC_VALUE IS (ui,ud,lo,hi,hz,wl,wh,lo_in,hi_in,hz_in,wl_in,
        wh_in,lo_out,hi_out,hz_out,wl_out,wh_out);
    FUNCTION VALUE_DICTIONARY(value:LOGIC_VALUE) RETURN EVENT_VALUE;
    CONSTANT PIN_CODES:STRING:="ux01zwlh-";
    END wav_logic;
PACKAGE BODY wav_logic IS
--***************************************************************
-- This VALUE_DICTIONARY is reasonably comprehensive. In general,
-- this function will suffice for most logic designs without
-- modification.
--***************************************************************
    FUNCTION VALUE_DICTIONARY(value:LOGIC_VALUE) RETURN EVENT_VALUE IS
        CONSTANT x_lo:EVENT_VALUE:= STATE= LOW AND STRENGTH= DRIVE;
        CONSTANT x_hi:EVENT_VALUE:= STATE= HIGH AND STRENGTH= DRIVE;
        CONSTANT x_hz:EVENT_VALUE:= STATE= MIDBAND AND
            STRENGTH= RESISTIVE;
        CONSTANT x_wl:EVENT_VALUE:= STATE= LOW AND STRENGTH= RESISTIVE;
        CONSTANT x_wh:EVENT_VALUE:= STATE= HIGH AND
            STRENGTH= RESISTIVE;
        BEGIN
        CASE value IS
            WHEN ui => RETURN STATE= UNKNOWN AND STRENGTH= UNKNOWN;
            WHEN ud => RETURN STATE= UNKNOWN AND STRENGTH= UNKNOWN;
            WHEN lo => RETURN x_lo;
            WHEN hi => RETURN x_hi;
            WHEN hz => RETURN x_hz;
            WHEN wl => RETURN x_wl;
            WHEN wh => RETURN x_wh;
            WHEN lo_in => RETURN x_lo AND DIRECTION= STIMULUS;
            WHEN hi_in => RETURN x_hi AND DIRECTION= STIMULUS;
            WHEN hz_in => RETURN x_hz AND DIRECTION= STIMULUS;
            WHEN wl_in => RETURN x_wl AND DIRECTION= STIMULUS;
            WHEN wh_in => RETURN x_wh AND DIRECTION= STIMULUS;
            WHEN lo_out => RETURN x_lo AND DIRECTION= RESPONSE;
            WHEN hi_out => RETURN x_hi AND DIRECTION= RESPONSE;
            WHEN hz_out => RETURN x_hz AND DIRECTION= RESPONSE;
            WHEN wl_out => RETURN x_wl AND DIRECTION= RESPONSE;
            WHEN wh_out => RETURN x_wh AND DIRECTION= RESPONSE;
            END CASE;
        END VALUE_DICTIONARY;
    END wav_logic;
```

Figure 7.4-1 continued. A WAVES Logic Value Package.

7.4.3.2 The function then employs a CASE statement in conjunction with these constants to assign the characteristics of each LOGIC_VALUE. It should be noted that only those characteristics applicable to a given LOGIC_VALUE element, need to be defined. It is not necessary to assign all parameters (i.e. STATE, STRENGTH, DIRECTION and RELEVANCE) to each LOGIC_VALUE element.

7.4.4 The PIN_CODES String

The PIN_CODES string is a list of those codes which will be used by the testbench to assign input signal levels and assess output responses. Following is the PIN_CODES declaration from the example "wav_logic" package.

CONSTANT PIN_CODES : STRING := "UX01ZWLH-";

7.4.4.1 In this example, the entire STD_LOGIC code set is employed for consistency with our model. It should be noted that the 'U', 'X', 'W' and '-' codes will all be mapped to the "ud" or undefined element in the VALUE_DICTIONARY. The user is free to assign any code string desired. An alternative approach might be to assign different codes for inputs and outputs (i.e. 'H', 'L' for inputs and '1', '0' for outputs).

7.4.4.2 Some WAVES references suggest that the PIN_CODES data type be assigned in the WAVES frames file. This will result in an inconsistency with the order of analysis defined for WAVES datasets unless package declarations and package bodies are analyzed separately. To circumvent this problem, the PIN_CODES string can be assigned in this file. The alternative here would be to leave the PIN_CODES declaration in the WAVES frame package declaration and analyze the WAVES frame package declaration before analyzing the body of the WAVES_INTERFACE package body.

7.4.5 Analysis

In the order of analysis, this package follows analysis of the WAVES_SYSTEM and WAVES_STANDARD packages. Once this package is analyzed, it is not necessary to reanalyze it unless it becomes necessary to change the LOGIC_VALUE or PIN_CODES data elements.

7.5 The WAVES Frame Package

Figure 7.5-1 contains an example of a WAVES frame package. Because of the manner in which the LOGIC_VALUE data type was assigned, with input, output and indeterminate values, it is appropriate to establish the FRAME_SET assignment functions in a similar fashion. The functions provided by this package are identified in Table 7.5-1. These functions will be used by the waveform generator procedure to schedule input and output events when a vector is applied to the model. The "simfrm" function will not be used in the example testbench.

Declaration	Subprogram Type
simfrm	function
simin	function
simout	function

Table 7.5-1. WAVES Frame Package Declarations.

7.5.1 Event Timing

For input events, the event time (etime) is set to 0 (simin function). Therefore, inputs are changed immediately in accordance with the input vector. On the other hand, the event time for output events (simout function) are delayed by 155 ns. This time is based upon (and is in excess of)the maximum propagation delay time for the modules being evaluated by the subject testbench.

7.5.2 Analysis

Once the WAVES logic value package has been analyzed, the WAVES_INTERFACE package is analyzed followed by the WAVES frame package.

7.6 The Device Under Test (DUT) Package.

The sole purpose of the DUT package is to declare the TEST_PINS data type. This data element is an enumerated type which identifies each pin in the module to be tested. Figure 7.6-1 contains an example DUT package for the DQD module example. Table 7.6-1 identifies type data type declared in this package.

7.6.1 Pin Order

```
*********************************************************************
--   (c) Copyright 1994 by the
--       Naval Air Warfare Center, Aircraft Division, Indianapolis
--   Source
--       Author(s):            Charles K. Rogers
--       Organization:           NAWC-ADI
--                             Code 306, MS-42
--                             6000 E 21st St
--                             Indianapolis,  IN  46219-2189
--                             Phone:   317-353-3579
--                             EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--       Reference:            MIL-M-28787/212
--       Project:              SHARP   TIREP
--   DESC Certification
--       Status:               TBD
--*******************************************************************
--   Revision History
--       Version:              1.0
--           Date:             16 May 1994
--           Comments:         Original Release
--*******************************************************************
--   Module Description
--       File:                 wav_frmp.wav
--       Module  Name(s):      wav_frame package
--       Constraints:          none
--       Limitations:          none
--       I/O  Format(s):       none
--       Purpose and Use:      This VHDL package provides the value dictionary
--           which is used in conjunction with the vector table.
--       Notes:                none
--*******************************************************************
--   Standard Libraries/Packages
--       Waves_Interface       waves interface package
--   Associated Packages (order of analysis implied)
--       wav_logic             simulator value dictionary package
--   Component  Models
--       none
--   Platform:                 486/33MHz PC
--   Software/Version:         V-System for Windows, Version 3.0
--*******************************************************************
--   Design Specification Elements
--       simfrm                function
--       simin                  function
--       simout                function
--*******************************************************************
LIBRARY waves;
USE waves.wav_logic.ALL;
USE waves.WAVES_INTERFACE.ALL;
PACKAGE wav_frame IS
    FUNCTION simfrm RETURN FRAME_SET;
```

Figure 7.5-1. A WAVES Frame Package.

```
    FUNCTION simin    RETURN FRAME_SET;
    FUNCTION simout RETURN FRAME_SET;
    END wav_frame;
PACKAGE BODY wav_frame IS
--*************************************************************
-- The following functions are based upon the PIN_CODES string being
-- set to "UX01ZWLH-". If a different PIN_CODES string is employed,
-- then these functions would need to be modified accordingly.
-- Otherwise, there are no changes required to this package.
--*************************************************************
    FUNCTION simfrm RETURN FRAME_SET IS
      BEGIN
      RETURN
        NEW_FRAME_SET('U',FRAME_EVENT(ui, ETIME(0   ns)))+
        NEW_FRAME_SET('X',FRAME_EVENT(ud, ETIME(0   ns)))+
        NEW_FRAME_SET('0',FRAME_EVENT(lo, ETIME(0   ns)))+
        NEW_FRAME_SET('1',FRAME_EVENT(hi, ETIME(0   ns)))+
        NEW_FRAME_SET('Z',FRAME_EVENT(hz, ETIME(0   ns)))+
        NEW_FRAME_SET('W',FRAME_EVENT(ud, ETIME(0   ns)))+
        NEW_FRAME_SET('L',FRAME_EVENT(wl, ETIME(0   ns)))+
        NEW_FRAME_SET('H',FRAME_EVENT(wh, ETIME(0   ns)))+
        NEW_FRAME_SET('-',FRAME_EVENT(ud, ETIME(0 ns))));
      END simfrm;
    FUNCTION simin RETURN FRAME_SET IS
      BEGIN
      RETURN
        NEW_FRAME_SET('U',FRAME_EVENT(ui, ETIME(0   ns)))+
        NEW_FRAME_SET('X',FRAME_EVENT(ud, ETIME(0   ns)))+
        NEW_FRAME_SET('0',FRAME_EVENT(lo_in, ETIME(0   ns)))+
        NEW_FRAME_SET('1',FRAME_EVENT(hi_in, ETIME(0   ns)))+
        NEW_FRAME_SET('Z',FRAME_EVENT(hz_in, ETIME(0   ns)))+
        NEW_FRAME_SET('W',FRAME_EVENT(ud, ETIME(0   ns)))+
        NEW_FRAME_SET('L',FRAME_EVENT(wl_in, ETIME(0   ns)))+
        NEW_FRAME_SET('H',FRAME_EVENT(wh_in, ETIME(0   ns)))+
        NEW_FRAME_SET('-',FRAME_EVENT(ud, ETIME(0 ns))));
      END simin;
    FUNCTION simout RETURN FRAME_SET IS
      BEGIN
      RETURN
        NEW_FRAME_SET('U',FRAME_EVENT(ui, ETIME(155   ns)))+
        NEW_FRAME_SET('X',FRAME_EVENT(ud, ETIME(155   ns)))+
        NEW_FRAME_SET('0',FRAME_EVENT(lo_out, ETIME(155   ns)))+
        NEW_FRAME_SET('1',FRAME_EVENT(hi_out, ETIME(155   ns)))+
        NEW_FRAME_SET('Z',FRAME_EVENT(hz_out, ETIME(155   ns)))+
        NEW_FRAME_SET('W',FRAME_EVENT(ud, ETIME(155   ns)))+
        NEW_FRAME_SET('L',FRAME_EVENT(wl_out, ETIME(155   ns)))+
        NEW_FRAME_SET('H',FRAME_EVENT(wh_out, ETIME(155   ns)))+
        NEW_FRAME_SET('-',FRAME_EVENT(ud, ETIME(155 ns))));
      END simout;
    END wav_frame;
```

Figure 7.5-1 continued. A WAVES Frame Package.

Declaration	Data Type
TEST_PINS	enumerated type

Table 7.6-1. Device Under Test Package Declarations.

```
--***************************************************************
--   (c) Copyright 1994 by the
--        Naval Air Warfare Center, Aircraft Division, Indianapolis
--   Source
--        Author(s):           Charles K. Rogers
--        Organization:          NAWC-ADI
--                             Code 306, MS-42
--                             6000 E 21st St
--                             Indianapolis,  IN  46219-2189
--                             Phone:   317-353-3579
--                             EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--        Reference:           MIL-M-28787/212
--        Project:             SHARP   TIREP
--   DESC Certification
--        Status:              TBD
--***************************************************************
--   Revision History
--        Version:             1.0
--             Date:           11 May 1994
--             Comments:       Original Release
--***************************************************************
--   Module Description
--        File:                dqd_dutp.wav
--        Module  Name(s):     package dqd_dut
--        Constraints:         none
--        Limitations:         none
--        I/O  Format(s):      none
--        Purpose and Use:     This VHDL package provides the test_pins
--             declaration for this design.
--        Notes:               none
--***************************************************************
--   Standard Libraries/Packages
--        none
--   Associated Packages (order of analysis implied)
--        none
--   Component  Models
--        none
--   Platform:                 486/33MHz PC
--   Software/Version:         V-System for Windows, Version 3.0
--***************************************************************
```

Figure 7.6-1. The DQD_DUT Package.

```
--    Design Specification Elements
--          TEST_PINS              enumerated type declaration
--***************************************************************
PACKAGE dqd_dut  IS
--***************************************************************
-- The TEST_PINS data type is composed of the port signal names for the
-- VHDL module to be tested by the testbench.    The order of the signals
-- appearing in this declaration must be the same as the order of the
-- signals in the supporting vector data file.
--***************************************************************
      TYPE TEST_PINS IS  (urc_s,stb_cs,srm,sow_s,sdi,rdy_s,p_u,ovr_0,
            not_stb_s,not_ovd_s,not_mclr_s,not_lrc_s,not_eoc1,not_eoc,
            not_dta_1_rz,not_dta_0_rz,not_act,ldl1,ldl,g,f,ena_0,e,
            dta_clk_s,dta_1_s,dta_0_s,d33,d32,d,cnt_2,cnt_1,cnt_0,clk1,clk,
            c,b,act,a,vcc,gnd);
END  dqd_dut;
```

Figure 7.6-1 continued. The DQD_DUT Package.

The order of the pins declared in the TEST_PINS data type, must be the same as their occurrence in the external vector file which will be discussed later.

7.6.2 Analysis

The DUT package is analyzed following the WAVES frame package.

7.7 The Waveform Generator Package

Figure 7.7-1 contains an example of a waveform generator package for use with the DQD module example. Table 7.7-1 contains a listing of the declarations and procedures defined by this package.

Declaration	Data Type
input_pins	enumerated type
output_pins	enumerated type

Declaration	Subprogram Type
dqdwav	procedure

Table 7.7-1. Waveform Generator Package Declarations.

7.7.1 Input and Output Pin Sets

The "input_pins" and "output_pins" data types are basically subsets of the TEST_PINS data type which was defined in the DUT package. The waveform generator procedure will use these pin assignments to distinguish between inputs and outputs in the vector file since the PIN_CODES used in this example, could apply to either.

7.7.2 The DQDWAV Procedure

The "dqdwav" procedure is the waveform generator procedure. This procedure will read vector and time information from the file "dqd_vec.dat". It will then utilize the appropriate APPLY procedure from the WAVES_OBJECTS package to generate appropriate stimulus and response information for the testbench.

7.7.3 Analysis

```
--*****************************************************************
--  (c) Copyright 1994 by the
--        Naval Air Warfare Center, Aircraft Division, Indianapolis
--  Source
--        Author(s):            Charles K. Rogers
--        Organization:           NAWC-ADI
--                              Code 306, MS-42
--                              6000 E 21st St
--                              Indianapolis,  IN  46219-2189
--                              Phone:   317-353-3579
--                              EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--        Reference:            MIL-M-28787/212
--        Project:              SHARP   TIREP
--  DESC Certification
--        Status:               TBD
--*****************************************************************
--  Revision History
--        Version:              1.0
--              Date:           16 May 1994
--              Comments:       Original Release
--*****************************************************************
--  Module Description
--        File:                 dqd_wavp.wav
--        Module  Name(s):      dqd_wav package
--        Constraints:          none
--        Limitations:          none
--        I/O  Format(s):       none
--        Purpose and Use:      This VHDL package provides the waveform
--              generator procedure for the DQD standard hardware module.
--        Notes:                none
--*****************************************************************
--  Standard Libraries/Packages
--         Waves_Interface      waves interface package
--         Waves_Objects        waves objects package
--         textio               standard text i/o package
--  Associated Packages (order of analysis implied)
--         wav_logic            simulator value dictionary package
--         wav_frame            simulator code map package
--          dqd_dut             WAVES test pin package
--          dqd_vec.dat         WAVES vector data file
--  Component  Models
--        none
--  Platform:                   486/33MHz PC
--  Software/Version:           V-System for Windows, Version 3.0
--*****************************************************************
--  Design Specification Elements
--        input_pins            constant
--        output_pins           constant
--        dqdwav                procedure
```

Figure 7.7-1. A Waveform Generator Package.

```
--          data_file                    file
--*****************************************************************
LIBRARY waves;
USE waves.wav_logic.ALL;
USE waves.WAVES_INTERFACE.ALL;
USE waves.wav_frame.ALL;
USE  work.dqd_dut.ALL;
USE work.WAVES_OBJECTS.ALL;
PACKAGE dqd_wav IS
--*****************************************************************
-- The following pinsets must be declared. These pinsets are used
-- to distinguish inputs from outputs in the vector file.
--*****************************************************************
   CONSTANT input_pins:PINSET:= NEW_PINSET ((a,b,c,d,e,F,g,cnt_1,cnt_2,
      not_ovd_s,not_mclr_s,dta_clk_s,not_eoc,ena_0,clk,ldl,not_stb_s,
      sow_s,sdi,d32,d33,p_u,vcc,gnd));
   CONSTANT output_pins:PINSET:= NEW_PINSET ((cnt_0,clk1,rdy_s,stb_cs,
      not_eoc1,act,not_act,not_dta_0_rz,not_dta_1_rz,dta_0_s,dta_1_s,
      not_lrc_s,ovr_0,srm,urc_s,ldl1));
--*****************************************************************
-- The procedure "dqdwav" is named specifically for the subject
-- design in this example. The procedure name must be changed in
-- both the package declaration and the package body.
--*****************************************************************
   PROCEDURE dqdwav (SIGNAL wpl:INOUT WAVES_PORT_LIST;
      VARIABLE fileend:OUT BOOLEAN);
   END dqd_wav;
USE std.TEXTIO.ALL;
PACKAGE BODY dqd_wav IS
   PROCEDURE dqdwav (SIGNAL wpl:INOUT WAVES_PORT_LIST;
      VARIABLE fileend:OUT BOOLEAN) IS
--*****************************************************************
-- The filename for the "data_file" needs to be changed here to
-- identify the vector file for the design.
--*****************************************************************
      FILE data_file:TEXT IS IN "dqd_vec.dat";
      VARIABLE x:FILE_SLICE:= NEW_FILE_SLICE;
      CONSTANT single_frame_set_array:FRAME_SET_ARRAY:=
         NEW_FRAME_SET_ARRAY(simin,input_pins)+
         NEW_FRAME_SET_ARRAY(simout,output_pins);
      VARIABLE td:TIME_DATA:= NEW_TIME_DATA(single_frame_set_array);
      BEGIN
      fileend:= FALSE;
      LOOP
         READ_FILE_SLICE(data_file,x);
         EXIT WHEN  x.end_of_file=TRUE;
         APPLY(wpl,x.codes.ALL,DELAY(x.fs_time),td);
         END LOOP;
      IF  x.end_of_file=TRUE THEN  fileend:=TRUE;
```

Figure 7.7-1 continued. A Waveform Generator Package.

```
            END IF;
        END dqdwav;
    END dqd_wav;
```

Figure 7.7-1 continued. A Waveform Generator Package.

Following analysis of the DUT package, the WAVES_OBJECTS package is analyzed in preparation for the analysis of the waveform generator package.

7.8 The Vector File

7.8 Figure 7.8-1 contains a section of a vector file for the DQD model example. This vector information was generated in accordance with the specification for this module (MIL-M-28787/212).

7.8.1 Comments

In the vector file, the percent (%) symbol denotes a comment. This file contains a header similar to the header provided in VHDL files. Additionally, this file contains a pin order identification which should replicate the TEST_PINS declaration in the DUT package.

7.8.2 Vector Symbols

The vectors in this file are comprised of the symbols defined in the PIN_CODES string. Any of these symbols are acceptable although only the '1' and '0' symbols are used in this example file.

7.8.3 Vector Timing

Each vector may be appended with a time duration value (i.e. 500 ns on the first vector). It is not necessary to include a time value for each vector. Once a vector time duration has been declared, that value will be used in subsequent vectors which do not contain a time value.

7.8.4 Potential Problem Areas

There are formatting elements of the vector file which are very sensitive to the testbench.

7.8.4.1 For example, there must be a space between the end of the vector and the colon (:) denoting a time field entry or semi-colon (;) denoting the end of the line. Additionally, there must be a space between the time value and the time units.

7.8.4.2 Also, care should be taken to insure that the symbols used in the vector file will be recognized by the testbench. The use of symbols which are not defined for the testbench should be avoided.

7.8.4.3 It is formatting problems related to this file which will account for many of the obscure errors generated through the WAVES standard packages when attempting to analyze and/or run a WAVES testbench.

7.9 The Testbench

Figure 7.9-1 contains the testbench for the DQD model example. It may be noted that the testbench entity declaration does not contain a port statement. The testbench is a self-contained entity which instantiates the device/assembly level model, applies stimulus patterns and evaluates responses. The only outputs from the testbench will be in the form of messages reported to the user describing the success or failure of a test.

7.9.1 Generics

For this example, the EDS model for the DQD circuit card assembly, described in Chapter 4 is used. The generic declaration for this module level model is carried up to the testbench allowing these elements to be redefined at runtime.

7.9.2 The Device Under Test

```
%*****************************************************************
%   (c) Copyright 1994 by the
%        Naval Air Warfare Center, Aircraft Division, Indianapolis
%   Source
%        Author(s):              Charles K. Rogers
%        Organization:             NAWC-ADI
%                                Code 306, MS-42
%                                6000 E 21st St
%                                Indianapolis,  IN  46219-2189
%                                Phone:   317-353-3579
%                                EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
%        Reference:              MIL-M-28787/212
%        Project:                SHARP   TIREP
%   DESC Certification
%        Status:                 TBD
%*****************************************************************
%   Revision History
%        Version:                1.0
%             Date:              16 May 1994
%             Comments:          Original Release
%*****************************************************************
%   Module Description
%        File:                   dqd_vec.dat
%        Module  Name(s):        none
%        Constraints:            none
%        Limitations:            none
%        I/O  Format(s):         none
%        Purpose and Use:        This VHDL package defines the functional vector
%             set for the DQD standard hardware module.
%        Notes:                  none
%*****************************************************************
%   Standard Libraries/Packages
%        none
%   Associated Packages (order of analysis implied)
%        none
%   Component  Models
%        none
%   Platform:                    486/33MHz PC
%   Software/Version:            V-System for Windows, Version 3.0
%*****************************************************************
%   Design Specification Elements
%        entire file
%*****************************************************************
% Pin order is
%urc_s,stb_cs,srm,sow_s,sdi,rdy_s,p_u,ovr_0,not_stb_s,not_ovd_s,
%not_mclr_s,not_lrc_s,not_eoc1,not_eoc,not_dta_1_rz,not_dta_0_rz,
%not_act,ldl1,ldl,g,f,ena_0,e,dta_clk_s,dta_1_s,dta_0_s,d33,d32,d,
```

Figure 7.8-1. The DQD Module Test Vector File,
a Partial Listing.

```
%cnt_2,cnt_1,cnt_0,clk1,clk,c,b,act,a,vcc,gnd
1000011000011111101111000100111100110110.500ns;
100001000001111110111100010011110011111,0
100001000101111110111100010011110011111,0
100001100101111110111100010011110011011,0
100000100110111110111101010011111011011,0
000000101011101110111101010011110011011,0
000000111110101110111101010011111011011,0
000001101101101110111101010011110011011,0
000001101110101110111101010011111011011,0
000001101111111110111100010011110011011,0
000001001111111100111100110011110011111,0
000001101101111110111100010011110011011,0
000001101111111110111100010011110011011,0
000001101111011110111110010011110011011,0
101000100111011110011011010011111011011,0
001000101111011110011011010011111011011,0
001001101101011110011011010011110011011,0
001001101111011110011011010011111011011,0
001001101111011110011011010011111111011,0
101000100111011110011011010011111111011,0
001000101111011110011011010011111111011,0
001001101101011110011011010011110111011,0
001001101111011110011011010011111111011,0
001001101111011110011011010011111011011,0
001001101101011110011110100111100110110
001001101101011110011110100111101110110
001001101111011110011110100111111110110
001001101111011110011110100111110110110
001001101101011110011110010011110011011,0
000001101101011110111010010011110111011,0
000001101101001110111011010011110111011,0
000001101110001110111011010011111111011,0
000001101110001000111011010011111011111,0
000001101110111000111011010001111011111,0
000001101110111000111011010011111001111,0
000001101110111000111011010011111010111,0
000001101110111000111011010011111011101,0
000001101110011000111011010011111011111,0
000001101110011000111011010011111111111,0
000001101110011000111011010011101011111,0
000001101110011000111011010010101011111,0
000001101110011000111011010011101011111,0
000001101110001000111011010011111011111,0
000001101110111000101011010011111011111,0
000001101110111000101011010011111111111,0
000001101110111000101011010011101011111,0
```

Figure 7.8-1 continued. The DQD Module Test Vector File, a Partial Listing.

```
--****************************************************************
--   (c) Copyright 1994 by the
--        Naval Air Warfare Center, Aircraft Division, Indianapolis
--   Source
--        Author(s):            Charles K. Rogers
--        Organization:           NAWC-ADI
--                              Code 306, MS-42
--                              6000 E 21st St
--                              Indianapolis,  IN  46219-2189
--                              Phone:   317-353-3579
--                              EMail:   ROGERSC1@po2.nawc-ad-indy.navy.mil
--        Reference:            MIL-M-28787/212
--        Project:              SHARP   TIREP
--   DESC Certification
--        Status:               TBD
--****************************************************************
--   Revision History
--        Version:              1.0
--             Date:            16 May 1994
--             Comments:        Original Release
--****************************************************************
--   Module Description
--        File:                 dqd_tstt.vhd
--        Module  Name(s):      dqd_tst/arch_waves_app_tb
--        Constraints:          none
--        Limitations:          none
--        I/O  Format(s):       none
--        Purpose and Use:      This VHDL module describes the testbench for the
--             DQD standard hardware module. This testbench was developed in
--             accordance with M28787/212.
--        Notes:                none
--****************************************************************
--   Standard Libraries/Packages
--        std_logic_1164        standard multi-value logic package
--        waves_objects         WAVES objects package
--   Associated Packages (order of analysis implied)
--        eia_567               standard eia package
--        dqd_ds                DQD module design specification package
--        wav_logic             simulator value dictionary package
--   Component  Models
--        dqd_eds               the electronic data sheet model
--   Platform:                  486/33MHz PC
--   Software/Version:          V-System for Windows, Version 3.0
--****************************************************************
--   Design Specification Elements
--        wire delays           generic map
--        operating points      generic map
--        dqd_eds               model under test component
--        w_signals             waveform generator stimulus/response signals
```

Figure 7.9-1. The DQD Module Testbench.

```
--      p_signals          model response signals
--      to_1164            function
--      translate          process
--      errchk             process
--*****************************************************************
LIBRARY ieee;
LIBRARY eia;
LIBRARY waves;
USE ieee.STD_LOGIC_1164.ALL;
USE waves.wav_logic.ALL;
USE work.WAVES_OBJECTS.ALL;
USE work.dqd_wav.ALL;
USE eia.EIA_567.ALL;
USE  work.dqd_ds.ALL;
ENTITY dqd_tst IS
--*****************************************************************
-- Following are the input and output routing delays defined for this
-- design. This listing should be altered to reflect the design being
-- modelled.
--*****************************************************************
   GENERIC(wi_a,wi_b,wi_c,wi_d,wi_e,wi_f,wi_g,wi_d32,wi_d33,wi_sow_s,
      wi_p_u,wi_ldl,wi_not_ovd_s,wi_dta_clk_s,wi_clk,wi_ena_0,
      wi_not_eoc,wi_not_mclr_s,wi_cnt_1,wi_cnt_2,wi_sdi,
      wi_not_stb_s,wi_act,wi_not_act,wi_cnt_0,wi_dta_0s,wi_dta_1s,wi_srm,
      wi_urc_s,wi_not_lrcs,wi_clk1,wi_ldl1,wi_rdy_s,wi_not_dta0rz,
      wi_not_dta1rz,wi_stb_cs,wi_ovr_0,wi_not_eoc1: TIME:=0  ns;
--*****************************************************************
-- Following is a standard set of operating points. This listing
-- needs to be updated if additional operating points are supported
-- by the design.
--*****************************************************************
-- Valid operating points are:
--     prop_del=tmin,  op_temp=-55  degrees_c,  op_volt=(5.5  v)
--     prop_del=tnom,  op_temp=27  degrees_c,  op_volt=(5  v)
--     prop_del=tmax,  op_temp=125  degrees_c,  op_volt=(4.5  v)
--     prop_del=tzero,  op_temp=27  degrees_c,  op_volt=(5  v)
--*****************************************************************
      prop_del: OPERATING_SELECTION:=  TNOM;
      op_temp: TEMPERATURE:=27   degrees_c;
      op_volt: VOLTAGE:=5  v;
--*****************************************************************
-- The x_generation global variable controls 'X' state generation by
-- the "async_checker" and "sync_checker" modules when an
-- assert condition is violated. If true, then 'X' states are generated
-- on assertion violations. If false, then 'X' states will not be
-- generated.
--*****************************************************************
         x_generation: BOOLEAN:=  FALSE;
--*****************************************************************
```

Figure 7.9-1 continued. The DQD Module Testbench.

```
-- The m_generation variable is used to control reports provided by
-- assertion statements in the "sync_checker" and "async_checker
-- modules. If the severity_level of the m_generation variable is less
-- than or equal to the serverity_level of an assertion violation, then
-- the assertion report will be generated. Otherwise, the report will
-- not be provided. Valid severity_level's are note, warning, error
-- or failure.
--***************************************************************
            m_generation: SEVERITY_LEVEL := FAILURE);
    END dqd_tst;
ARCHITECTURE arch_waves_app_tb OF dqd_tst IS
--***************************************************************
-- The model under test will need to be declared as a component
-- and instantiated in the body of the testbench.
--***************************************************************
    COMPONENT dqd_eds
        GENERIC(wi_a,wi_b,wi_c,wi_d,wi_e,wi_f,wi_g,wi_d32,wi_d33,wi_sow_s,
            wi_p_u,wi_ldl,wi_not_ovd_s,wi_dta_clk_s,wi_clk,wi_ena_0,
            wi_not_eoc,wi_not_mclr_s,wi_cnt_1,wi_cnt_2,wi_sdi,
            wi_not_stb_s,wi_act,wi_not_act,wi_cnt_0,wi_dta_0s,wi_dta_1s,
            wi_srm,wi_urc_s,wi_not_lrcs,wi_clk1,wi_ldl1,wi_rdy_s,
            wi_not_dta0rz,wi_not_dta1rz,wi_stb_cs,wi_ovr_0,
            wi_not_eoc1: TIME:=0 ns;
            user_operating_point: POINT;
            x_generation: BOOLEAN;
            m_generation: SEVERITY_LEVEL);
        PORT(a,b,c,d,e,f,g,d32,d33,sow_s,p_u,ldl,not_ovd_s,dta_clk_s: IN STD_LOGIC;
            clk,ena_0,not_eoc,not_mclr_s,cnt_1,cnt_2,sdi,not_stb_s: IN STD_LOGIC;
            act,not_act,cnt_0,dta_0_s,dta_1_s,srm,urc_s,not_lrc_s,clk1: INOUT STD_LOGIC;
            ldl1,rdy_s,not_dta_0_rz,not_dta_1_rz,stb_cs,ovr_0,not_eoc1: INOUT STD_LOGIC;
            vcc,gnd:IN STD_LOGIC);
        END COMPONENT;
    FOR ALL:dqd_eds USE ENTITY work.dqd_eds;
--***************************************************************
-- The following signals are used for stimulus/response information
-- provided from the vector file through the waveform generator.
-- These signals are based upon the design being tested.
--***************************************************************
    SIGNAL w_a,w_b,w_c,w_d,w_e,w_f,w_g,w_d32,w_d33,w_sow_s,w_p_u,w_ldl,
        w_not_ovd_s,w_dta_clk_s,w_clk,w_ena_0,w_not_eoc,w_not_mclr_s,
        w_cnt_1,w_cnt_2,w_sdi,w_not_stb_s,w_act,w_not_act,w_cnt_0,
        w_dta_0_s,w_dta_1_s,w_srm,w_urc_s,w_not_lrc_s,w_clk1,w_ldl1,w_rdy_s,
        w_not_dta_0_rz,w_not_dta_1_rz,w_stb_cs,w_ovr_0,w_not_eoc1,
        w_vcc,w_gnd:STD_LOGIC;
--***************************************************************
-- The following signals are used for response information from the
-- model being tested. These signals are required for each of the
-- model outputs.
--***************************************************************
```

Figure 7.9-1 continued. The DQD Module Testbench.

```
SIGNAL    p_act,p_not_act,p_cnt_0,p_dta_0_s,p_dta_1_s,p_srm,p_urc_s,
      p_not_lrc_s,p_clk1,p_ldl1,p_rdy_s,p_not_dta_0_rz,p_not_dta_1_rz,
      p_stb_cs,p_ovr_0,p_not_eoc1: STD_LOGIC;
SIGNAL waves_data:WAVES_PORT_LIST;
SIGNAL errcnt:INTEGER:=0;
SIGNAL filend:BOOLEAN;
CONSTANT  user_operating_point:POINT:=(selection=>prop_del,temp=>op_temp,
      SUPPLY=>(0=>op_volt));
--*****************************************************************
-- It is only necessary to change this function if a different
--  VALUE_DICTIONARY  is  used.
--*****************************************************************
FUNCTION to_1164 (value:LOGIC_VALUE) RETURN STD_LOGIC IS
      BEGIN
      CASE value IS
            WHEN ui  =>  RETURN'U';
            WHEN ud  =>  RETURN'X';
            WHEN lo  =>  RETURN'0';
            WHEN hi  =>  RETURN'1';
            WHEN hz  =>  RETURN'Z';
            WHEN wl  =>  RETURN'L';
            WHEN wh  =>  RETURN'H';
            WHEN lo_in  =>  RETURN'0';
            WHEN hi_in  =>  RETURN'1';
            WHEN hz_in  =>  RETURN'Z';
            WHEN wl_in  =>  RETURN'L';
            WHEN wh_in  =>  RETURN'H';
            WHEN lo_out  =>  RETURN'0';
            WHEN hi_out  =>  RETURN'1';
            WHEN hz_out  =>  RETURN'Z';
            WHEN wl_out  =>  RETURN'L';
            WHEN wh_out  =>  RETURN'H';
            END CASE;
      END to_1164;
BEGIN
--*****************************************************************
-- The waveform generator procedure name needs to be altered to
-- correspond with the design being tested.
--*****************************************************************
waves:PROCESS
      VARIABLE fileend:BOOLEAN;
      BEGIN
      dqdwav(waves_data,fileend);
      filend<=fileend;
      WAIT;
      END PROCESS;
--*****************************************************************
-- The translate process needs to be altered to reflect the signals
-- provided through the waveform generator.
```

Figure 7.9-1 continued. The DQD Module Testbench.

```
--*****************************************************************
    translate: PROCESS(waves_data)
        BEGIN
        w_urc_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(1).l_value));
        w_stb_cs  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(2).l_value));
        w_srm  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(3).l_value));
        w_sow_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(4).l_value));
        w_sdi  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(5).l_value));
        w_rdy_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(6).l_value));
        w_p_u  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(7).l_value));
        w_ovr_0  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(8).l_value));
        w_not_stb_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(9).l_value));
        w_not_ovd_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(10).l_value));
        w_not_mclr_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(11).l_value));
        w_not_lrc_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(12).l_value));
        w_not_eoc1  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(13).l_value));
        w_not_eoc  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(14).l_value));
        w_not_dta_1_rz  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(15).l_value));
        w_not_dta_0_rz  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(16).l_value));
        w_not_act  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(17).l_value));
        w_ldl1  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(18).l_value));
        w_ldl  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(19).l_value));
        w_g  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(20).l_value));
        w_f  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(21).l_value));
        w_ena_0  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(22).l_value));
        w_e  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(23).l_value));
        w_dta_clk_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(24).l_value));
        w_dta_1_s   <=to_1164(LOGIC_VALUEval(waves_data.wpl(25).l_value));
        w_dta_0_s  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(26).l_value));
        w_d33  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(27).l_value));
        w_d32  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(28).l_value));
        w_d  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(29).l_value));
        w_cnt_2  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(30).l_value));
        w_cnt_1  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(31).l_value));
        w_cnt_0  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(32).l_value));
        w_clk1  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(33).l_value));
        w_clk  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(34).l_value));
        w_c  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(35).l_value));
        w_b  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(36).l_value));
        w_act  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(37).l_value));
        w_a  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(38).l_value));
        w_vcc  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(39).l_value));
        w_gnd  <=  to_1164(LOGIC_VALUEval(waves_data.wpl(40).l_value));
        END PROCESS;
--*****************************************************************
-- The model under test will need to be declared as a component
-- and instantiated in the body of the testbench.
--*****************************************************************
    z1:dqd_eds  GENERIC MAP(wi_a,wi_b,wi_c,wi_d,wi_e,wi_f,wi_g,wi_d32,
```

Figure 7.9-1 continued. The DQD Module Testbench.

```
            wi_d33,wi_sow_s,wi_p_u,wi_ldl,wi_not_ovd_s,wi_dta_clk_s,wi_clk,
            wi_ena_0,wi_not_eoc,wi_not_mclr_s,wi_cnt_1,wi_cnt_2,wi_sdi,
            wi_not_stb_s,user_operating_point,x_generation,m_generation)
        PORT MAP(w_a,w_b,w_c,w_d,w_e,w_f,w_g,w_d32,w_d33,w_sow_s,w_p_u,
            w_ldl,w_not_ovd_s,w_dta_clk_s,w_clk,w_ena_0,w_not_eoc,
            w_not_mclr_s,w_cnt_1,w_cnt_2,w_sdi,w_not_stb_s,p_act,p_not_act,
            p_cnt_0,p_dta_0_s,p_dta_1_s,p_srm,p_urc_s,p_not_lrc_s,p_clk1,
            p_ldl1,p_rdy_s,p_not_dta_0_rz,p_not_dta_1_rz,p_stb_cs,p_ovr_0,
            p_not_eoc1,w_vcc,w_gnd);
--*****************************************************************
-- The following process monitors the output signals provided from
-- the waveform generator. This process is triggered by a transaction
-- on any of these signals. When triggered, the process will compare
-- compare all outputs from the model with the expected outputs from
-- the vector file. Any errors will be logged. Alter this process to
-- operate on the corresponding output signals for the design being
-- tested.
--*****************************************************************
    errchk: PROCESS(w_act'transaction,w_not_act'transaction,
        w_cnt_0'transaction,w_dta_0_s'transaction,w_dta_1_s'transaction,
        w_srm'transaction,w_urc_s'transaction,w_not_lrc_s'transaction,
        w_clk1'transaction,w_ldl1'transaction,w_rdy_s'transaction,
        w_not_dta_0_rz'transaction,w_not_dta_1_rz'transaction,
        w_ovr_0'transaction,w_stb_cs'transaction,w_not_eoc1'transaction,
        filend)
        BEGIN
        IF filend THEN
            IF errcnt0 THEN
                ASSERT FALSE REPORT "testbench failed" SEVERITY FAILURE;
                ELSE ASSERT FALSE REPORT "testbench passed" SEVERITY NOTE;
                END IF;
            ELSIF now0 ns THEN
            ASSERT w_act=p_act REPORT "error in act" SEVERITY WARNING;
            ASSERT w_not_act=p_not_act REPORT "error in not_act"
                SEVERITY WARNING;
            ASSERT w_cnt_0=p_cnt_0 REPORT "error in cnt_0" SEVERITY WARNING;
            ASSERT w_dta_0_s=p_dta_0_s REPORT "error in dta_0_s"
                SEVERITY WARNING;
            ASSERT w_dta_1_s=p_dta_1_s REPORT "error in dta_1_s"
                SEVERITY WARNING;
            ASSERT w_srm=p_srm REPORT "error in srm" SEVERITY WARNING;
            ASSERT w_urc_s=p_urc_s REPORT "error in urc_s" SEVERITY WARNING;
            ASSERT w_not_lrc_s=p_not_lrc_s REPORT "error in not_lrc_s"
                SEVERITY WARNING;
            ASSERT w_clk1=p_clk1 REPORT "error in clk1" SEVERITY WARNING;
            ASSERT w_ldl1=p_ldl1 REPORT "error in ldl1" SEVERITY WARNING;
            ASSERT w_rdy_s=p_rdy_s REPORT "error in rdy_s" SEVERITY WARNING;
            ASSERT w_not_dta_0_rz=p_not_dta_0_rz REPORT "error in not_dta_0_rz"
                SEVERITY WARNING;
```

Figure 7.9-1 continued. The DQD Module Testbench.

```
            ASSERT  w_not_dta_1_rz=p_not_dta_1_rz  REPORT "error in not_dta_1_rz"
                SEVERITYWARNING;
            ASSERT  w_stb_cs=p_stb_cs  REPORT "error in stb_cs"
                SEVERITYWARNING;
            ASSERT  w_ovr_0=p_ovr_0  REPORT "error in ovr_0" SEVERITYWARNING;
            ASSERT  w_not_eoc1=p_not_eoc1  REPORT "error in not_eoc1"
                SEVERITYWARNING;
            IF  w_act/=p_act  OR  w_not_act/=p_not_act  OR  w_cnt_0/=p_cnt_0  OR
                w_dta_0_s/=p_dta_0_s   OR  w_dta_1_s/=p_dta_1_s  OR  w_srm/=p_srm  OR
                w_urc_s/=p_urc_s   OR  w_not_lrc_s/=p_not_lrc_s  OR  w_clk1/=p_clk1  OR
                w_ldl1/=p_ldl1  OR  w_rdy_s/=p_rdy_s  OR
                w_not_dta_0_rz/=p_not_dta_0_rz   OR  w_not_dta_1_rz/=p_not_dta_1_rz  OR
                w_stb_cs/=p_stb_cs  OR  w_ovr_0/=p_ovr_0  OR  w_not_eoc1/=p_not_eoc1
                THEN     errcnt<=errcnt+1;
                END IF;
            END IF;
        END PROCESS;
    END  arch_waves_app_tb;
```

Figure 7.9-1 continued. The DQD Module Testbench.

The "dqd_eds" component is declared and instantiated in the testbench architecture as "z1". This is the only component in the testbench.

7.9.3 Internal Signals

A set of internal signals are declared which provide for connection between the testbench and the "dqd_eds" component. Additionally, signals are declared which will hold the expected responses from the external vector file for comparison with the responses generated by the "dqd_eds" component.

7.9.1.4 The 'to_1164"Function

A "to_1164" function is declared which maps LOGIC_VALUE states to STD_LOGIC states. In this case, this function is based upon the "UX01Z subtype of STD_LOGIC.

7.9.1.5 Reading Data

Within the "dqd_eds" architecture, the waveform generator (dqdwav) is invoked. This procedure will read a vector from the external data file and return it as "waves_data".

7.9.1.6 Data Translation

The "translate" process which is sensitive to "waves_data" will map the vector to the internal signal set for application to the "dqd_eds" component and for verification of the "dqd_eds" responses.

7.9.1.7 The Data Monitor

The final process of this testbench is sensitive to events on the internal response signals. It may be recalled that these response signals are delayed by 155 ns from the beginning of the frame (see paragraph 7.5.1). At the time of an event on one of these signals, all signals from the "dqd_eds" component should be stable, allowing this process to compare the "dqd_eds" component responses with the expected responses as defined in the external vector file.

7.9.1.8 Errors

Anytime an error is detected between expected and received responses, a warning message is generated. Once all vectors have been applied (i.e. the end of the external vector file has been reached), a message will be provided which indicates the success or failure of the test. If one or more errors is encountered, the number of errors will be so noted. Under the approach presented, multiple errors on a single vector will be reported as a single error.

7.9.2 Analysis

The testbench file is the last file to be analyzed for a given model.

This page intentionally left blank