March 25, 1994

VITAL

VHDL Initiative Toward ASIC Libraries

Model Development Specification

Version v2.2b

Table of Contents

1.0 Introduction

The objectives of the VITAL initiative can be summed up in one sentence:

> "Accelerate the development of sign-off quality ASIC macrocell simulation libraries written in VHDL by leveraging existing methodologies of model development"

1.1 Background

The VITAL effort germinated from ideas generated at the VHDL International Users' Forum held in Scottsdale AZ in May 1992. Further discussions brought people to the conclusion that the biggest impediment to VHDL design was the lack of ASIC libraries; and that the biggest impediment to ASIC libraries was the lack of a uniform, efficient method for handling timing in VHDL. Since this problem had already been solved for other languages it was clear that a solution in VHDL was possible and that an effective way to arrive at this solution was to leverage existing technology.

VITAL obtains its leverage from (a) the Standard Delay File (SDF) format, in addition to (b) certain contributed elements of the Std_Timing package of the VHDL Technology Group and specialized timing and behavioral techniques provided by Ryan & Ryan, and (c) the existence of numerous ASIC libraries and tools developed using SDF timing implementations. Leveraging existing tools and environments is viewed as being a catalyst for the rapid deployment of ASIC libraries once this initiative is standardized under the IEEE.

1.2 VITAL Organization

The VHDL Initiative Towards ASIC Libraries (VITAL) is an industry-based, informal consortium formed with the following in mind:

- **Charter**: Accelerate the availability of ASIC libraries across industry VHDL simulators.
- **Objectives**: (1) High-performance, accurate (sign-off quality) ASIC simulation across VITAL-compliant EDA tools from a single ASIC vendor description, and (2) aggressive time-to-market.
- **Approach**: Define a modeling specification (in conjunction with VHDL packages) that leverages existing practices and techniques, is compliant with IEEE Standards 1076 and 1164, and utilizes OVI's SDF timing format. Standardize the approved result through the IEEE.
- **End-Product:** (1) VITAL_Timing VHDL package defining standard, acceleratable timing procedures for delay value selection, timing checks, and timing error reporting, (2) VITAL_Primitives VHDL package defining standard, acceleratable primitives for boolean and table-based functional description, (3) Specification of OVI's Standard Delay Format (SDF) for communication of instance delay values and (4) Model Development Specification document defining utilization of VITAL and VHDL elements for ASIC library development.

1.3 VITAL Steering Committee

A steering committee consisting of members from The VHDL Technology Group, Ryan&Ryan, Synopsys, Cadence, Hewlett-Packard and Texas Instruments provides the day-to-day operation of the VITAL program.

1.4 VITAL Membership

Over 45 companies have participated in the development of the VITAL technical plans. They have been led by William D. Billowitch of the VHDL Technology Group who serves as technical director; Steve Schulz of Texas Instruments' Defense Systems and Electronics Group who serves as U.S. coordinator.

1.5 Purpose

The purpose of this document is to provide

1. The definition of a model style standard which is described as "VITAL compliant". This style guide forms the body of this document.
2. The definition of a VITAL_TIMING package which includes timing-related subprograms and definitions.
3. The definition of a VITAL_PRIMITIVES package which contains combinatorial primitives as well as Truth and State tables.
4. The definition of a mapping of SDF 2.0 syntax / constructs to Generic parameter values on VITAL models.

The information presented in this document is the result of numerous discussions with ASIC vendors, CAE tool vendors, and ASIC designers to determine the requirements for effective design and fabrication of ASICs using VHDL. The specification particularly addresses the highest priority issues singled out by this group. These are:

- Timing accuracy
- Model maintainability
- Simulation performance

1.6 Scope

This specification is targeted toward VHDL logic simulation modeling of ASIC cells at a level of accuracy sufficient for sign-off.

The techniques employed in this specification could be readily applied toward the broader segment of VHDL modeling including FPGA, standard component models and systems.

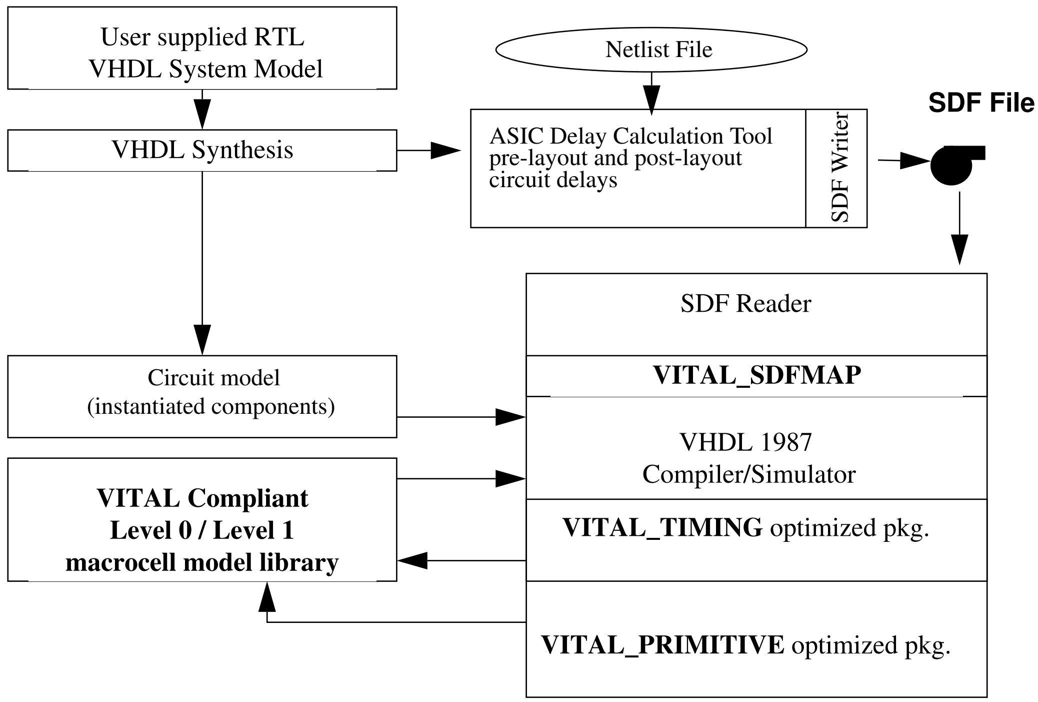
1.7 "Compliance" with this specification

Simulators will be considered VITAL compliant if they correctly implement the SDF mapping and the packages outlined in this specification.

This document, and the package declarations define the standard. This document and the package body VHDL code define a simulatable specification of the semantics of the package. Vendors may choose to implement the package bodies as they see fit, provided the functionality matches that of the specified bodies.

1.8 VITAL Elements

The VITAL initiative has defined the five areas of standardization which are used to define what are called “VITAL compliant models” and “VITAL compliant simulators”.



VITAL_TIMING: A package containing data types and subprograms to support development of macrocell timing models. Included in this package are routines for delay selection, timing violation checking and reporting, and glitch detection.

VITAL_PRIMITIVE: A set of commonly used combinatorial primitives are defined in this package. The primitives are provided both in Function and concurrent Procedure form to support either behavioral or structural modelling styles. The procedure primitives support separate pin-to-pin delay path and well as GlitchOnEvent glitch detection. In addition, this package contains general purpose Truth and State Tables.

VITAL_SDFMAP: The SDF to VHDL mapping specification defines the mapping / translation of SDF data to Generic Parameter values on VITAL models. Simulation vendors will use this mapping standard to build tools which automatically back-annotate the VHDL circuit with timing data values.

VITAL compliant Level 0 / Level 1 Models: The main body of this specification defines a style for writing VITAL compliant models.

The VITAL_Timing and VITAL_Primitives package semantics are defined by their VHDL description according to the IEEE 1076-87 LRM. Their interfaces are defined by the package declaration and their functionality is defined by the package body. Implementors are ecnourage to optimize the performance of these packages as long as the semantics of the implementation agree with those defined by the VHDL description.

2.0 VITAL Model Interface Specification (Level 0)

VITAL models are defined by an interface specification which provides a standardization of the names of ports and generics such that back-annotation via Standard-Delay-File (SDF) mapping can be accomplished. Models which adhere to this interface specification are considered to be "VITAL Level 0 Complaint".

Furthermore, VITAL Level 0 compliant models may be written to follow the VITAL Model Architecture Specification in the following section. The architecture specification illustrates how to utilize the VITAL_Timing and VITAL_Primitives packages in order to achieve performance improvements and VITAL Level 1 Compliance.

2.1 General Philosophy

The VITAL model interface is characterized by the following major considerations:

Functionality and Timing Semantics Defined ONLY within the VHDL model: Certain HDL's allow the semantic of timing information to be decoupled from the functionality of the model. While this technique provides the modeler and simulator with certain desirable flexibilities at times, it also leaves open the potential for a functional-timing mismatch to occur. In this VITAL specification, all timing and all functionality shall be defined within the VHDL model and its associated VHDL packages.

Standard Delay File (SDF) Mapping: SDF will be used solely as a provider of timing data values, and not the semantic for timing cause and effect. It is expected, although not required, that CAE vendors will provide a direct method for reading SDF information into the design hierarchy. Certain rules are set forth in this specification which govern how SDF timing data values are mapped to the VHDL design hierarchy.

Timing Calculations Performed Off Chip: With the exception of the use of VITAL_Timing routines, all timing calculations, shall be performed "off-chip". All load dependent or environmentally dependent timing data shall be calculated outside of the VITAL model and then be provided to the VITAL model solely as actual values to the model's generic parameter list or via SDF direct import. This methodology has the benefit of relieving the model from the computational load of timing parameter determination thereby simplifying the model, improving its runtime performance and lowering its' memory utilization.

Interoperability: VITAL models shall be written to produce the same functionality and same timing behavior regardless of the simulator on which the VITAL model is exercised.

2.2 Design Unit Naming Specification

Entity, Architecture, and Configuration names shall be limited to 28 characters.

Note: Certain simulators use the name of the design unit as a name for a subdirectory created in the compiled library. While most UNIX systems have no restriction of naming length, experience has found that, many times, these subdirectories cross over Network File System (NFS) mount points. An often forgotten limitation is that NFS supports files of atomic length less than 28. (i.e. /mynode/supercalifragilisticexpialadoshious is too long but /mynode/I_am_a_28_character_filename is acceptable). TAR file archive software also exhibits problems with file names in excess of 28 characters.

2.3 Signal Specifications

- Ports shall use the Std_logic_1164 data type or subtypes. [IEEE Std 1164].
- Scalar Ports: To provide a consistent interface for I/O signals, models shall use the std_logic datatype or a subtype thereof for scalar ports.
- Array Ports: Models shall use the std_logic_vector datatype. No subtypes are allowed.

Optimization note: Simulators may be able to recognize the existence of resolved signals which do not actually have multiple drivers. In such cases, the processing of the Std_logic_1164 resolution function need not be applied since the resolution function is well known and easily optimized.

- All signals shall use Std_Logic_1164 data types.

2.4 Use of Attributes in Level 0

- Level 0 compliant models shall not make use of vendor supplied attributes or other non-normative VHDL constructs such as meta-comments or directives in such a manner as to affect the function or timing characteristics of the model.

2.5 Back-Annotation Specification

- If timing information is required for a VITAL model, then the sole point of entry of timing information into a VITAL model shall be via the model's generic parameter formal list.
- All timing parameter determinations (including min/typ/max delay selection) shall be performed "off-chip".
- All back-annotation information shall be passed to the model via generic parameters.

2.6 Identifier naming Recommendation (not req'd for Vital compliance)

With the exception of generic parameters, it is recommended that developers utilize a naming convention where multi-word names are built by capitalizing the first letter of each word and abutting each word together with its neighbor without the use of special characters such as "_".

Recommended Style: ThisIsAnInvertedSignalNameN

Discouraged Style: this_is_an_inverted_signal_name

2.7 Generic Parameter Specifications

VITAL generic parameters follow a strict set of rules. These rules must be observed in order to back-annotate timing information into VITAL models. A mapping mechanism is established (see Appendix C) between SDF information and the names of ports and generic parameters on VITAL models. Successful mapping relies upon the SDF names matching the names of the generic parameters. Should this name association be broken, back-annotation will not be possible via the SDF mapping techniques specified in this document.

2.7.1 Generic *Timing* Parameter Types

All back-annotated Generic timing parameters shall be of one of the following TYPEs. These TYPEs are declared in the package 'VITAL_Timing'.

```
TYPE TransitionType        is ( tr01, tr10, tr0z, trz1, tr1z, trz0 );
TYPE TransitionArrayType   is ARRAY (TransitionType range <>) of TIME;

SUBTYPE DelayTypeXX           is TIME;
SUBTYPE DelayType01           is TransitionArrayType (tr01 to tr10);
SUBTYPE DelayType01Z          is TransitionArrayType (tr01 to trz0);
TYPE    DelayArrayTypeXX      is ARRAY (natural range <>) of DelayTypeXX;
TYPE    DelayArrayType01      is ARRAY (natural range <>) of DelayType01;
TYPE    DelayArrayType01Z     is ARRAY (natural range <>) of DelayType01Z;
TYPE    TimeArray             IS ARRAY ( natural range <> ) OF TIME;
```

2.7.2 Generic parameter naming specification

Back-Annotation mapping from SDF to the VITAL model requires that a named mapping be established between the model's generic parameters and the SDF information. VITAL Level 0 compliant models shall adhere to the naming conventions specified in this document.

The naming conventions of a Generic timing parameter communicate:

- the kind of timing parameter (e.g. propagation delay, setup time)
- the port(s) or delay path(s) for which the parameter applies.

The TYPE associated with a generic timing parameter communicates the form of the timing value (e.g. single value, state dependent value list, etc.)

```
tpd_CLK_Q : DelayTypeXX            := 5 ns;
tpd_CLK_Q : DelayType01            := (tr01 => 2 ns, tr10 => 3 ns);
tpd_CLK_Q : DelayType01Z           := ( 1 ns, 2 ns, 3 ns, 4 ns, 5 ns, 6 ns );
tpd_CLK_Q : DelayArrayTypeXX(0 to 1):= (0 => 5 ns,
                                        1 => 6 ns);
tpd_CLK_Q : DelayArrayType01(0 to 1):= (0 => (tr01 => 2 ns, tr10 => 3 ns),
                                        1 => (tr01 => 2 ns, tr10 => 3 ns));
tpd_CLK_Q : DelayArrayType01Z(0 to 1):= (0 => ( 1 ns, 2 ns, 3 ns, 4 ns, 5 ns, 6 ns ),
                                          1 => ( 1 ns, 2 ns, 3 ns, 4 ns, 5 ns, 6 ns ));
```

There are no required timing parameters for a VITAL model.

2.7.3 Generic Timing Parameter Prefixes

VITAL generic timing parameters are of one of the following forms:

- <VITAL_prefix>
- <VITAL_prefix>_<port1-name>
- <VITAL_prefix>_<port1-name>_<port2-name>_<condition>_<edge1>_<edge2>

The following VITAL timing parameter prefixes are defined:

<VITAL-prefix> ::=

tipd	|	Interconnect Path Delay (IPD)
tpd	|	propagation delay
tsetup	|	setup constraint
thold	|	hold constraint
trelease	|	release constraint (like setup for asynch signals)
trecovery	|	recovery constraint (like hold for asynch signals)
tperiod	|	period where min or max is not specified
tperiod_min	|	minimum period
tperiod_max	|	maximum period
tpw	|	minimum pulsewidth
tdevice	|	indicates to which subcomponent the spec. applies
tskew	|	indicates to which subcomponent the spec. applies
tpulse		for path pulse delay

NOTE: The classification of the clock edge as a 'rising' or 'falling' edge is not specified by a time parameter. It is assumed to be implied by the behavior of the model.

- These generic parameters shall use the specific type declarations from the VITAL_timing package.

Definition: In the following discussion, an Input Port is any port of mode IN or INOUT. An Output Port is any port of mode OUT, INOUT, or BUFFER.

Terms:

Propagation Delay - The time delay from the arrival of an input signal value to the appearance of a corresponding output signal value.

Setup Time - The time period prior to a clock edge during which the specified input signal value may not change value.

Hold Time - The time period following a clock edge during which the specified input signal value may not change value.

Release Time - A change to an unasserted value on the specified asynchronous (set, reset) input signal must precede the clock edge by at least the release time.

Removal Time - An asserted condition must be present on the specified asynchronous (set, reset) input signal for at least the removal time following the clock edge.

Period - The time delay from the leading edge of a clock pulse to the leading edge of the following clock pulse.

tpd .. . . Propagation Delay

The **tpd** prefix is used to denote Propagation Delay parameters.

Allowed parameter form(s):

- tpd . . . . delay applies to ANY input-to-output path of the model
- tpd_<port1-name> . . . . delay applies to all delay paths to the indicated OUTPUT port.
- tpd_<port1-name>_<port2-name> . . . . delay applies to ONLY the specified INPUT-to-OUTPUT delay path.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayType01 .. . . Two delay values (tr01, tr10)
- DelayType01Z . . . . Six delay values (tr01, tr10, tr0z, trz1, tr1z, trz0)
- DelayArrayTypeXX . . . . Array of delays (vector input)
- DelayArrayType01 . . . . Array of delay pairs
- DelayArrayType01Z. . . . . Array of delay 6-tuples

tsetup . . . . Input Setup Time

The **tsetup** prefix is used to denote Setup Time parameters.

Allowed parameter form(s):

- tsetup_<port1-name> . . . . The setup time applies to any INPUT port <port1-name> with respect to any clock signal on the model.
- tsetup_<port1-name>_<port2-name> . . . . The setup time applies to *the* INPUT port <port1-name> with respect to the clock signal <port2-name>.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayType01 .. . . Two delay values
- DelayArrayTypeXX . . . . Array of delays (vector input)
- DelayArrayType01 . . . . Array of delay pairs (vector input)

thold . . . . Input Hold Time

The thold prefix is used to denote Hold Time parameters.

Allowed parameter form(s):

- thold_<port1-name> . . . . The hold time applies to *the* INPUT port named <port1-name> with respect to any clock signal on the model.
- thold_<port1-name>_<port2-name> . . . . The hold time applies to *the* INPUT port named <port1-name> with respect to the clock signal <port2-name>.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayType01 .. . . Two delay values
- DelayArrayTypeXX . . . . Array of delays (vector input)
- DelayArrayType01 . . . . Array of delay pairs (vector input)

trelease . . . . Input Release Time

The **trelease** prefix is used to denote Release Time parameters.

Allowed parameter form(s):

- trelease_<port1_name> . . . . The release time applies to an INPUT port named <port1_name> with respect to any clock signal on the model.
- trelease_<port1_name>_<port2_name> . . .. . . The release time applies to an INPUT port named <port1_name> with respect to the clock signal <port2_name>.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayArrayTypeXX . . . . Array of delays (vector input)

tperiod . . . . Period

The **tperiod** prefix is used to denote minimum and maximum Period Time parameters.

Allowed parameter form(s):

- tperiod_min_<port1-name> .. . . Minimum allowable period constraint applies to the port whose name is <port1-name>.
- tperiod_max_<port1-name>.. . . . . Maximum allowable period constraint applies to the port whose name is <port1-name>.
- tperiod_<port1_name>_<EdgeSpecifier> . .. . . Where EdgeSpecifier is either posedge or negedge.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayArrayTypeXX . . . . Array of delays (vector input)

trecovery . . . . Input Removal Time

The **tremoval** prefix is used to denote Removal Time parameters.

Allowed parameter form(s):

- **trecovery**_<port1-name> . . .. The recovery time applies to the INPUT port named <port1-name> with respect to any clock signal on the model.
- **trecovery**_<port1-name>_<port2-name> . .. . . The recovery time applies to the INPUT port named <port1-name> with respect to the clock signal <port2-name>.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayArrayTypeXX . . . . Array of delays (vector input)

tpw.. . . . . Pulse Width

The **tpw** prefix is used to denote minimum and maximum Pulse Width parameters.

The pulse width of a 0->1 or 1->0 transition of a periodic signal.

Allowed parameter form(s):

- tpw_hi_min_<port1-name> .. . . Minimum allowable time that <port1-name> must be held high for a pulse on <port1-name>
- tpw_hi_max_<port1-name> .. . . Maximum allowable time that <port1-name> can be held high.
- tpw_lo_min_<port1-name> .. . . Minimum allowable time that <port1-name> must be held low for a pulse on <port1-name>
- tpw_lo_max_<port1-name> .. . . Maximum allowable time that <port1-name> can be held low.
- tpw_<port1_name>_<EdgeSpecifier> . . . . Where EdgeSpecifier is either posedge or negedge.

Allowed parameter type(s):

- DelayTypeXX. . . . . Single delay value
- DelayType01 .. . . Two delay values (high,low)
- DelayArrayTypeXX . . . . Array of delays (vector input)

tipd . . . . Interconnect Path Delay (NETDELAY and INTERCONNECT delay)

- tipd_<portname> . . . . wire delay to / from that port to its interconnection point.

2.8 Other Generic Parameters

Developers may include additional generic parameters to control functionality.

2.8.1 Generic Parameter Control Switches

The following Generic Control Parameter names and types are defined. These parameters are not required, however, models should use these parameters over any other arbitrary parameter names. VITAL support tools may provide special support for these parameter names. Models should not use these parameter name for other purposes.

```
InstancePath  : string; -- This may be done implicitly or via 1076-93 VHDL
--------------------------------------------------------------
-- Switches
--------------------------------------------------------------
TimingChecksOn   Boolean := TRUE; -- if TRUE, then timing checks are enabled.
XGenerationOn    Boolean := TRUE; -- if TRUE, then timing or functional violations
                                    cause certain output ports to be assigned an 'X'.
```

2.8.2 Use of Back-Annotation Generic Parameters inside of the model

- Back-Annotation generic parameters shall not be used in any context which requires that the VALUE of the generic parameter be known prior to the start of the initialization phase for simulation. Specifically: Back-Annotation generic parameters should be considerd as having an unknown value during elaboration.Back-Annotation generic parameters shall not be used as default parameters within the model.
- Back-Annotation generic parameters may be used as actuals associated with formal parameters of subprograms called from the VITAL_Timing package.
- Back-Annotation generic parameters may be used to declare a DelayType01Z typed timing constant which obtains its value solely by assignment of the return value of the VitalExtendToFillDelay subprogram.
- Back-Annotation generic parameters can be associated with GENERIC parameters on component instances within the model.

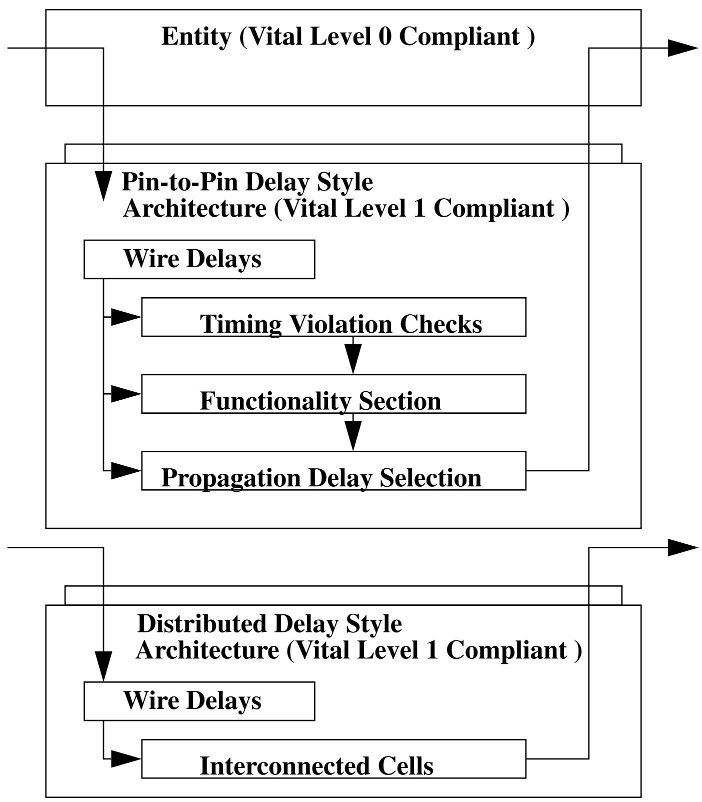
2.9 Example VITAL Level 0 Model Interface

```
-- ----------------------------------------------------------------------
--   File name :  DLE24.vhd
-- ----------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- ----------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V0.001   |  rjr   |  02/20/94  | Initial detailed coding
-- ----------------------------------------------------------------------
LIBRARY IEEE;         USE IEEE.std_logic_1164.ALL;
LIBRARY VITAL;        USE VITAL.VITAL_timing.all;
                      USE VITAL.VITAL_primitives.all;
------------------------------------------------------------------------
-- ENTITY declaration
------------------------------------------------------------------------
ENTITY DLE24 IS
    GENERIC (
              tipd_A           : DelayType01 := (0 ns, 0 ns);
              tipd_CLK         : DelayType01 := (0 ns, 0 ns);
              tipd_CLRZ        : DelayType01 := (0 ns, 0 ns);
              tipd_RL          : DelayType01 := (0 ns, 0 ns);
              tpd_A_Y_MODE_EQ_1: DelayType01 := (1 ns, 1 ns);
              tpd_A_Y_MODE_EQ_0: DelayType01 := (1 ns, 1 ns);
              tpd_CLK_Q1       : DelayType01 := (1 ns, 1 ns);
              tpd_CLK_Q0       : DelayType01 := (1 ns, 1 ns);
              tpd_CLK_MODE     : DelayType01 := (1 ns, 1 ns);
              tpd_CLK_CLKS     : DelayType01 := (1 ns, 1 ns);
              tpd_CLRZ_Q1      : DelayType01 := (1 ns, 1 ns);
              tpd_CLRZ_Q0      : DelayType01 := (1 ns, 1 ns);
              tpd_CLRZ_MODE    : DelayType01 := (1 ns, 1 ns);
              tpd_CLRZ_CLKS    : DelayType01 := (1 ns, 1 ns);
              tsetup_RL_CLK    : DelayType01 := (1 ns, 1 ns);
              thold_RL_CLK     : DelayType01 := (1 ns, 1 ns);
              tsetup_CLRZ_CLK  : DelayType01 := (1 ns, 1 ns);
              thold_CLRZ_CLK   : DelayType01 := (1 ns, 1 ns);
              tperiod_CLK      : DelayTypeXX := (10 ns);
              tpw_CLK          : DelayType01 := (5 ns, 5 ns);
              TimingCheckOn    : BOOLEAN := TRUE;
              InstancePath     : STRING := "*" );
    PORT    ( A     : IN    std_logic := 'U';
              CLK   : IN    std_logic := 'U';
              CLRZ  : IN    std_logic := 'U';
              RL    : IN    std_logic := 'U';
              Y     : OUT   std_logic;
              CLKS  : OUT   std_logic;
              MODE  : OUT   std_logic;
              Q0    : OUT   std_logic;
              Q1    : OUT   std_logic  );
 END DLE24;
```

3.0 VITAL Model Architectural Specification (Level 1)

Models written to the VITAL Level 0 Specification should be capable of handling back-annotation via SDF mapping. However, the amount of time spent simulating large ASIC designs with unaccelerated models may prove problematic, particularly as design sizes increase.

Level 1 provides a usage model for constructing complete cell models from the building blocks provided by Level 0. By following these guidelines the cell models will be constrained to the point where it will facilitate simulator vendors to optimize the set-up and execution of the models leading to higher levels of performance than can be expected through accelerating the building block packages alone.



3.1 Attributes for Level 1 Optimization

The VITAL_LEVEL1 attribute shall be set TRUE in the architecture so that Level 1 compliance can be verified.

The following attribute has been declared to signify that a given architecture is Level 1 compliant.

ATTRIBUTE VITAL_LEVEL1 : Boolean;

Example:

```
ATTRIBUTE VITAL_LEVEL1 of LEVEL_1 : architecture is TRUE;
```

3.2 Level 1 Architectural Declarations

The only objects which can be declared in a Level 1 architecture are internal signals.

EXAMPLE:

```
-- ---------------------------------------------------------------------
--   File name :  <EntityName>.vhd
-- ---------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- ---------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V0.001   |  me    |  02/20/94  | Initial detailed coding
-- ---------------------------------------------------------------------
LIBRARY IEEE;         USE IEEE.std_logic_1164.ALL;
LIBRARY VITAL;        USE VITAL.VITAL_timing.all;
                      USE VITAL.VITAL_primitives.all;
------------------------------------------------------------------------
-- ENTITY declaration
------------------------------------------------------------------------
ENTITY <EntityName> IS
    GENERIC (
               VitalPrefix_<portname> : DelayType01 := (0 ns, 0 ns);
               ...
               TimingCheckOn    : BOOLEAN := TRUE;
               InstancePath     : STRING := "*" );

    PORT    ( Inport1     : IN     std_logic := 'U';
               ...
              InOutPortN : INOUT  std_logic := 'U';
               ...
              Outport1    : OUT   std_logic);
END <EntityName>;

ARCHITECTURE Level1Behavioral OF <EntityName> IS

    -- declare one internal signal for each IN or INOUT port
    SIGNAL Inport1_ipd    : std_logic := 'X';
    SIGNAL InOutPortN_ipd : std_logic := 'X';


BEGIN
```

3.3 Interconnect Path Delay Handling

Simple wire delays may be modeled within the WIRE_DELAY block using the following techniques:

- For each generic parameter indicating a wire delay specification (i.e. tipd_<portname>), declare a signal in the architecture (e.g. SIGNAL <portname>_ipd) which will serve as the internal, post delayed signal.
- If the input signal to be delayed is a vector, a generate statement shall be used inside the WIRE_DELAY block, so as to delay each subelement of the vector signal.
- In the generate block, there shall be no declarations and the only statement should be a concurrent procedure call to VitalPropagateWireDelay().
- If the tipd_* generic is not of type DelayType01Z, then the predefined type-conversion function VItalExtendToFillDelay may be used.
- Within the declared WIRE_DELAY block, propagate the wire delay by utilizing the VitalPropagateWireDelay () concurrent procedure. There shall be no other statements in the block other than the concurrent procedure calls or a generate loop.
- Outside this block, for every delayed signal (*_ipd signal), there shall be no reference to the undelayed input signal. In all such cases, the corresponding *_ipd signal should be used.

EXAMPLE:

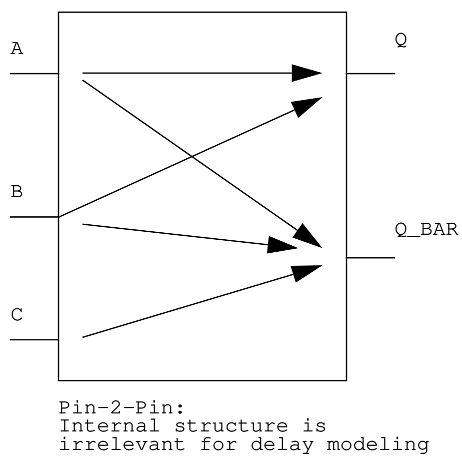
```
-----------------------------------------------------------
-- This block is OPTIONAL. Used for modeling wire delays.
-----------------------------------------------------------
WIRE_DELAY : block
begin
    -- one concurrent procedure call for each IN or INOUT signal
   VitalPropagateWireDelay (      OutSig => <portname>_ipd,
                                  InSig  => <portname>,
                                  twire  => tipdz_<portname>  );
    -- Port delays for a vector input of DATASIZE
    gen_data: for I in databus'range generate
         VitalPropagateWireDelay( outsig => databus_ipd(i),
                                  insig  => databus(i),
                                  twire  => VitalDelay_twire_databus);
    end generate;

   -- certain IN or INOUT ports will be associated with negative hold
   -- time constraints. In order to properly handle negative hold time
   -- constraints, the affected port is delayed by the hold time.
   VitalPropagateWireDelay (      OutSig => <portname>_ipd,
                                  InSig  => <portname>,
                                  twire  => tipdz_<portname>,
                                  t_hold_hi => thold_<port1>_<port2>,
                                  t_hold_lo => thold_<port1>_<port2>
                             );
end block;
```

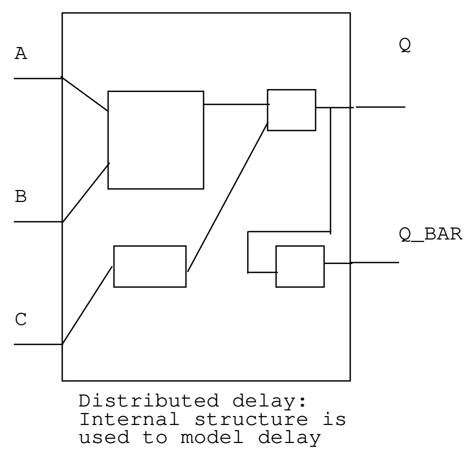
3.4 Level-1 Distributed Delay Modeling

Distributed delay is a style of delay modeling where an ASIC cell is actually composed of structural portions, each is left to account for its own delay. The output of the ASIC cell is an artifact of the structure, event profile and actual delays that are present in the underlying implementation of this cell. In Figure 1. the structural differences between Distributed and Pin-to-Pin delay models are illustrated.

Functionality for the Distributed Delay model (or portion thereof) shall be contained within the VITAL_-NETLIST block.

- Only calls to VITAL_Primitives concurrent procedures may be made within this block.
- Local signals may be declared solely to support the interconnection of the concurrent procedures.
- Timing information may be passed into the concurrent procedures from the generic parameter list.





Rule of Implementation:

- A given ASIC cell shall use either the Distributed Delay modeling style or the Pin-to-Pin Delay style but not both in the same model.

VITAL_NETLIST Distributed Delay Model Example:

```
-- SAMPLE ASIC CELL WHICH USES VITAL LEVEL-1 GUIDELINES
-- This model uses VITAL primitives in the distributed delay style.
-- The device delays are passed as generics which can be back-annotated.

LIBRARY IEEE;
USE     IEEE.Std_logic_1164.all;
LIBRARY VITAL;
USE     VITAL.VITAL_Timing.all;
use     VITAL.VITAL_Primitives.all;


entity AO2 is
    generic (tdevice_A1_Q : DelayType01 := (10 ns, 11 ns); -- Device delay for A1
             tdevice_A2_Q : DelayType01 := (10 ns, 11 ns); -- Device delay for A2
             tdevice_O1_Q : DelayType01 := (11 ns, 12 ns));-- Device delay for O1
    port (Y : OUT std_logic; A, B, C, D : IN std_logic);
end;

architecture Level1DistributedDelay of AO2 is
    -- This attribute indicates that model is VITAL LEVEL1 compliant
    attribute VITAL_LEVEL1 of Level1DistributedDelay : architecture is TRUE;
begin
    ------------------------------------------------------------
    -- This block is used for Distributed delay modelling.
    ------------------------------------------------------------
    VITAL_NETLIST : block
        SIGNAL A1Out, A2Out : std_logic; -- local signals only
    begin
         -- Concurrent procedure calls to VITAL_Primitives
        A1: VitalAND2 ( q => A1Out, a => A, b => B,
                        tpd_a_q => tdevice_A1_Q,
                        tpd_b_q => tdevice_A1_Q);
        A2: VitalAND2 ( q => A2Out, a => C, b => D,
                        tpd_a_q => tdevice_A2_Q,
                        tpd_b_q => tdevice_A2_Q);
        O2: VitalOR2  ( q => Y, a => A1Out, b => A2Out,
                        tpd_a_q => tdevice_O1_Q,
                        tpd_b_q => tdevice_O1_Q);
    end block;
end;
```

3.5 Level 1 Pin-to-Pin Delay Modeling

Functionality for the Pin-to-Pin Delay model (or portion thereof) shall be contained within the VITALBehavior process. n the interest of acceleration, only routines found in the VITAL_Timing and VITAL_Primitives packages, or overloaded operators in the std_logic_1164 package [IEEE Std 1164] shall be used.

- In addition to the WIRE_DELAY block, one process is mandatory in the cell architecture. This process describes the function & timing of the cell. This process is labeled: **VitalBehavior** and is divided into three parts: Timing Checks, Functionality, path delay selection. The following are some general rules that apply:
- The VITALBehavior process has no wait statements, and the sensitivity list includes a reference to all the input ports or their *_ipd signal counterparts, but not both.
- The structure of the process is rigid! It requires the timing check section followed by the Functionality section, followed by the path delay selection section with no other code in between them.
- No control structures are allowed (if, case, loop) in the process, with the exception of one 'if' statement in the timing checks section.
- No signal assignments are allowed (outside of the path delay section), and only to the output.
- Data is passed between sections in the process using variables. Specifically, the violation section is only allowed to make assignments to violation flags.

EXAMPLE:

```
------------------------------------------------------------
-- This Process is used for the Pin-to-Pin delay model.
-- If this model contains only distributed delay concurrent
-- VITAL_Primitive procedure calls, then this Process shall not
-- be used. Otherwise it is mandatory. This process expresses
-- the functionality and timing of the cell.
------------------------------------------------------------
-- sensitivity clause is to all internally delayed (*_ipd) signals.
VITALBehavior : PROCESS ( <portname>_ipd, <portname>_ipd, ...)
-- Possible declarations for intermediate variables or constants
-- for example, violation flags, or other state variables, see detail.
begin
```

3.6 Additional Level 1 Pin-to-Pin Delay Modeling Rules

- All functionality of a model should always be described in a process named ***VitalBehavior***, for acceleration purposes.
- No other process or blocks are used in the architecture.
- TimingCheck:, Functionality: and PathDelay: labels shall be inserted where appropriate to indicate the first line of each section of the VitalBehavior process.

3.6.1 Timing Checks Section

The Timing Check section consists of a single 'if' statement using the 'TimingChecksOn' generic as the only condition. When the TimingChecksOn parameter is TRUE, the statements within the 'if' statement execute. [Therefore, if TimingChecks are not desired during a given simulation run, substantial speed improvements can be realized.] The model designer has the responsibility of assuring proper model's functionality even if the generic parameter 'TimingChecksOn' is FALSE.

The 'then' part of the if statement shall contain only calls to VITAL_Timing sub-programs. No 'else' part is allowed. In addition the following rules are recognized:

- Each timing check will return a std_logic variable the value of which will be interpreted to mean that a violation has occurred and was detected.
- The timing checks shall not drive the output signal nor shall they drive any other signal.
- Outside of printing error messages, the timing check routines will not cause any other side effect.

EXAMPLE:

```
--------------------------------------------------------
-- Timing Checks Section
--------------------------------------------------------
TimingCheck: If (TimingChecksOn) then
    -- do timing checks on *_ipd signals
    -- the 'output' of this section are violation flags
    -- one per violation check.
end if;
```

3.6.2 Functionality Section

The Functionality section may contain:

- one or more calls to subprograms contained in the VITAL_Primitives package.
- assignments to internal temporary variables.

EXAMPLE:

```
 ---------------------------------------------------------------
 -- Functionality Section
 --------------------------------------------------------------
Functionality: violation := Tviol_<FirstTest> OR
                Tviol_<SecondTest> OR ...
                Tviol_<LastTest>;
-- call to VITAL_Primitives subprogram which determine the
-- output with the timing violations considered as an additional
-- input parameter

VitalStateTable (
     StateTable => DLE24_tab,
     DataIn => ( violation, CLK_ipd, CLRZ_CLK_ipd, RL_CLK_ipd),
     NumStates => 5,
     Result => Results,
     PreviousDataIn => PrevData );

  Y_zd := To_X01(A_ipd);
```

3.6.3 Path Delay Selection Section

Path delays are handled in this section of the model. Each output signal will be driven by a call to VitalPropagatePathDelay(...).

It is the job of the VitalPropagatePathDelay(...) routine to determine the appropriate delay to use given that an output signal has changed value. It accomplishes this task by knowing when the input Paths(...) to the routine last changed their values. From that information, it picks the minimum delay of all possible active stimulus-response paths.

Scalar Example:

```
--------------------------------------------------------
-- Pin-to-Pin Delay Section
--------------------------------------------------------
-- One call to VitalPropagatePathDelay for each output signal.
PathDelay: VitalPropagatePathDelay(
    OutSignal     => Q2,       -- Actual OUTPUT signal
    OutSignalName => "Q2",
    OutTemp       => Q2_zd,    -- Zero delayed internal Output signal
    -- First of two input pins which affect the output Q2
    Paths(0).InputChangeTime => Clock_ipd'last_event,
    Paths(0).PathDelay       => VitalExtendToFillDelay(tpd_clock_q2),
    Paths(0).Condition       => (RESET = '0'),
    -- Second of two input pins which affect the output Q2
    Paths(1).InputChangeTime => Data_ipd'last_event,
    Paths(1).PathDelay       => VitalExtendToFillDelay(tpd_data_q2),
    Paths(1).Condition       => (RESET = '0'),
    GlitchData    => GlitchData,
    GlitchMode    => MessagePlusX,
    GlitchKind    => OnEvent);
```

Rules

For the parameters to the path delay procedure, "explicit" rules are hereby specified so as to avoid any potential misuse of this procedure (thereby adversely affecting optimizations).:

- OutSignal must be an output/inout port
- OutTemp must be output_zd
- Paths.InputChangeTime must be a 'LAST_EVENT of input or *_ipd signals
- Parameters GlitchMode and Glitchking should be locally static
- GlitchData should not be used in ANY other expression

Vector Signal Example:

```
---------------------------------------------------------
-- Pin-to-Pin Delay Section
---------------------------------------------------------
DataBusPaths : For i in databus'range LOOP
    VitalPropagatePathDelay(
    OutSignal     => Databus(i),
    OutSignalName => "Databus(...)",
    OutTemp       => Databus_zd(i),
    -- First of two input pins which affect the output
    Paths(0).InputChangeTime => Clock_ipd'last_event,
    Paths(0).PathDelay       => VitalExtendToFillDelay(tpd_clock_databus),
    Paths(0).Condition       => (RESET = '0'),
    -- Second of two input pins which affect the output
    Paths(1).InputChangeTime => Iack_ipd'last_event,
    Paths(1).PathDelay       => VitalExtendToFillDelay(tpd_iack_databus),
    Paths(1).Condition       => (RESET = '0'),
    GlitchData    => GlitchDataArray(i),
    GlitchMode    => MessagePlusX,
    GlitchKind    => OnEvent);
end Loop;
```

3.7 Level 1 Example Architecture

```
ARCHITECTURE Behavior OF DLE24 IS

    SIGNAL A_ipd     : std_ulogic := 'X';
    SIGNAL CLK_ipd   : std_ulogic := 'X';
    SIGNAL CLRZ_ipd  : std_ulogic := 'X';
    SIGNAL RL_ipd    : std_ulogic := 'X';

    SIGNAL CLRZ_CLK_ipd    : std_ulogic := 'X';
    SIGNAL RL_CLK_ipd      : std_ulogic := 'X';

BEGIN

  --------------------
  -- INPUT PATH DELAYs
  --------------------
  WIRE_DELAY : BLOCK
  BEGIN
    VitalPropagateWireDelay (    A_ipd,    A, tipdz_A   );
    VitalPropagateWireDelay (  CLK_ipd,  CLK, tipdz_CLK );
    VitalPropagateWireDelay ( CLRZ_ipd, CLRZ, tipdz_CLRZ);
    VitalPropagateWireDelay (   RL_ipd,   RL, tipdz_RL  );
    VitalPropagateWireDelay ( CLRZ_CLK_ipd, CLRZ, tipdz_CLRZ, thold_CLRZ_CLK);
    VitalPropagateWireDelay (   RL_CLK_ipd,   RL, tipdz_RL  , thold_RL_CLK  );
  END BLOCK;

  --------------------
  -- Behavior Section
  --------------------
  VITALBehavior : PROCESS ( A_ipd, CLK_ipd, CLRZ_CLK_ipd, RL_CLK_ipd)

    -- Timing Check results
    VARIABLE Tviol_RL_CLK     : X01 := '0';
    VARIABLE Tviol_CLRZ_CLK   : X01 := '0';
    VARIABLE Pviol_CLK        : X01 := '0';
    VARIABLE PeriodCheckInfo_CLK : DelayArrayTypeXX(0 TO 1) := PeriodCheckInfo_Init;

    VARIABLE violation : X01 := '0';
    VARIABLE  PrevData : std_logic_vector(1 TO 4) := (OTHERS=>'X');
    VARIABLE   Results : std_logic_vector(1 TO 7) := (OTHERS=>'X');
    VARIABLE      Y_zd : std_ulogic;
    ALIAS Q0_zd   : std_ulogic IS results(4);
    ALIAS Q1_zd   : std_ulogic IS results(5);
    ALIAS MODE_zd : std_ulogic IS results(6);
    ALIAS CLKS_zd : std_ulogic IS results(7);

    VARIABLE Y_GlitchData    : GlitchDataType;
    VARIABLE CLKS_GlitchData : GlitchDataType;
    VARIABLE MODE_GlitchData : GlitchDataType;
    VARIABLE Q0_GlitchData   : GlitchDataType;
    VARIABLE Q1_GlitchData   : GlitchDataType;

    CONSTANT L : VitalTableSymbolType := '0';
```

```
    CONSTANT H : VitalTableSymbolType := '1';
    CONSTANT x : VitalTableSymbolType := '-';
    CONSTANT S : VitalTableSymbolType := 'S';
    CONSTANT R : VitalTableSymbolType := '/';
    CONSTANT U : VitalTableSymbolType := 'X';
    CONSTANT V : VitalTableSymbolType := 'B'; -- Valid clock signal (non-rising)

    CONSTANT DLE24_tab : VitalStateTableType (1 TO 16, 1 TO 16) := (
    -----------------------------------------------------------------
    --Viol CLK CLRZ RL   R  C0  C1  Q0  Q1   R  C0  C1  Q0  Q1 MODE CLKS
    -----------------------------------------------------------------
      ( U,  x,  x,  x,  x,  x,  x,  x,  x,  U,  U,  U,  U,  U,  U,  U ),

      ( x,  x,  L,  H,  x,  x,  x,  x,  x,  H,  L,  L,  L,  L,  L,  L ),
      ( x,  x,  L,  L,  x,  x,  x,  x,  x,  L,  L,  L,  L,  L,  L,  L ),

      ( x,  R,  H,  L,  L,  L,  x,  x,  x,  L,  H,  S,  S,  S,  S,  S ),
      ( x,  R,  H,  L,  L,  H,  L,  x,  x,  L,  L,  H,  S,  S,  S,  S ),
      ( x,  R,  H,  L,  L,  H,  H,  x,  x,  L,  L,  L,  S,  S,  S,  S ),
      ( x,  R,  H,  L,  L,  H,  H,  L,  x,  L,  L,  L,  H,  S,  H,  S ),
      ( x,  R,  H,  L,  L,  H,  H,  H,  L,  L,  L,  L,  L,  H,  L,  H ),
      ( x,  R,  H,  L,  L,  H,  H,  H,  H,  L,  L,  L,  L,  L,  L,  L ),

      ( x,  R,  H,  H,  H,  L,  x,  x,  x,  H,  H,  S,  S,  S,  S,  S ),
      ( x,  R,  H,  H,  H,  H,  L,  x,  x,  H,  L,  H,  S,  S,  S,  S ),
      ( x,  R,  H,  H,  H,  H,  H,  x,  x,  H,  L,  L,  S,  S,  S,  S ),
      ( x,  R,  H,  H,  H,  H,  H,  L,  x,  H,  L,  L,  H,  S,  H,  S ),
      ( x,  R,  H,  H,  H,  H,  H,  H,  L,  H,  L,  L,  L,  H,  L,  H ),
      ( x,  R,  H,  H,  H,  H,  H,  H,  H,  H,  L,  L,  L,  L,  L,  L ),

      ( x,  V,  x,  x,  x,  x,  x,  x,  x,  S,  S,  S,  S,  S,  S,  S ) );

  BEGIN
    --------------------
    -- Timing Check Section
    --------------------
    TImingCheck: IF (TimingCheckON) THEN
        -- setup RL high or low before rising CLK
        --  hold RL high or LOW after  rising CLK
      VitalTimingCheck ( RL_CLK_ipd, "RL",  CLK_ipd, "CLK",
                         tsetup_RL_CLK(tr01), tsetup_RL_CLK(tr10),
                         thold_RL_CLK(tr01), thold_RL_CLK(tr10),
                         (CLK_ipd='1'),
                         InstancePath & "/DLE24", Tviol_RL_CLK );
        -- setup CLRZ high before rising CLK
        --  hold CLRZ low  after  rising CLK
      VitalTimingCheck ( CLRZ_CLK_ipd, "CLRZ",  CLK_ipd, "CLK",
                         tsetup_CLRZ_CLK(tr01), 0 ns               ,
                         0 ns                 , thold_CLRZ_CLK(tr10),
                         (CLK_ipd='1'),
                         InstancePath & "/DLE24", Tviol_CLRZ_CLK );
        -- Pulse width, period check CLK
      VitalPeriodCheck ( CLK_ipd, "CLK",
                         tperiod_CLK, TIME'HIGH,
```

```
                         tpw_CLK(tr01), TIME'HIGH, tpw_CLK(tr10), TIME'HIGH,
                         PeriodCheckInfo_CLK, Pviol_CLK,
                         InstancePath & "/DLE24", TRUE );
    END IF; -- Timing Check Section
--------------------
-- Functionality Section
--------------------
Functionality: violation := Tviol_RL_CLK OR Tviol_CLRZ_CLK OR Pviol_CLK;
VitalStateTable ( StateTable => DLE24_tab,
                          DataIn => ( violation, CLK_ipd, CLRZ_CLK_ipd, RL_CLK_ipd),
                       NumStates => 5,
                          Result => Results,
                  PreviousDataIn => PrevData );
Y_zd := To_X01(A_ipd);
----------------------
-- Path Delay Section
----------------------
PathDelay: VitalPropagatePathDelay ( Y, "Y", Y_zd,
         Paths(0)=>( A_ipd    'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_A_Y_MODE_EQ_0), MODE_zd = '0' ),
         Paths(1)=>( A_ipd    'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_A_Y_MODE_EQ_1) , MODE_zd /= '0' ),
         GlitchData=>Y_GlitchData, GlitchMode=>Xonly, GlitchKind=>OnEvent );
    VitalPropagatePathDelay ( CLKS, "CLKS", CLKS_zd,
         Paths(0)=>( 0 ns                , ZeroDelay01Z   , (violation = 'X')),
         Paths(1)=>( CLK_ipd  'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLK_CLKS)   , (CLRZ_CLK_ipd /= '0') ),
         Paths(2)=>( CLRZ_ipd 'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLRZ_CLKS)  , (CLRZ_CLK_ipd /= '1') ),
         GlitchData=>CLKS_GlitchData, GlitchMode=>Xonly, GlitchKind=>OnEvent );
    VitalPropagatePathDelay ( MODE, "MODE", MODE_zd,
         Paths(0)=>( 0 ns                , ZeroDelay01Z   , (violation = 'X')),
         Paths(1)=>( CLK_ipd  'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLK_MODE) , (CLRZ_CLK_ipd /= '0') ),
         Paths(2)=>( CLRZ_ipd 'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLRZ_MODE)  , (CLRZ_CLK_ipd /= '1') ),
         GlitchData=>MODE_GlitchData, GlitchMode=>Xonly, GlitchKind=>OnEvent );
    VitalPropagatePathDelay ( Q0, "Q0", Q0_zd,
         Paths(0)=>( 0 ns                , ZeroDelay01Z   , (violation = 'X')),
         Paths(1)=>( CLK_ipd  'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLK_Q0)     , (CLRZ_CLK_ipd /= '0') ),
         Paths(2)=>( CLRZ_ipd 'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLRZ_Q0)    , (CLRZ_CLK_ipd /= '1') ),
         GlitchData=>Q0_GlitchData, GlitchMode=>Xonly, GlitchKind=>OnEvent );
    VitalPropagatePathDelay ( Q1, "Q1", Q1_zd,
         Paths(0)=>( 0 ns                , ZeroDelay01Z   , (violation = 'X')),
         Paths(1)=>( CLK_ipd  'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLK_Q1) , (CLRZ_CLK_ipd /= '0') ),
         Paths(2)=>( CLRZ_ipd 'LAST_EVENT,
         VitalExtendToFillDelay(tpdz_CLRZ_Q1), (CLRZ_CLK_ipd /= '1') ),
         GlitchData=>Q1_GlitchData, GlitchMode=>Xonly, GlitchKind=>OnEvent );
END PROCESS;
END Behavior;
```