1.0 Appendix A: VITAL_TIMING package

The VITAL_TIMING package provides a number of routines useful in the development of sign-off accurate timing models in VHDL. The routines apply themselves to three areas of a model's development:

Interconnect Delay Modeling

- VitalPropagateWireDelay . . . . handles wire delays

Timing Violation Checking

- VitalTimingCheck . . . . Checks for Setup and Hold Violations
- VitalReportSetupHoldViolation . . . . Reports a setup/hold violation
- VitalReportRlseRmvlViolation . . . . Reports a release/removal violation
- VitalSetupHoldCheck . . . . Check Setup & Hold times
- VitalPeriodCheck . . . . Checks for Periodicity Violations

Pin-to-Pin Delay Handling

- VitalExtendToFillDelay . . . . Extends delays to 6-element format
- VitalCalcDelay . . . . Calculate Output Delays
- VitalPropagatePathDelay . . . . Assigns conditional path delays
- VitalGlitchOnEvent . . . . Handles Glitch Propagation on Events
- VitalGlitchOnDetect . . . . Handles Glitch Propagation on Detection

1.1 Standardization Issues

a. The VITAL_Timing package shall be compiled into the VITAL library.

b. The standard for this package consists of (a) the text of this document, (b) the declarations of the VITAL_Timing package and (c) the semantics of the VHDL code provided in the VITAL_Timing package body. In the event of a conflict or misunderstanding of interpretations, the order of precendence just described shall prevail.

1.2 Common Parameters:

The following parameters appear as formal parameters of the subsequent subprograms.

- TestPort . . . .::= A SIGNAL parameter which represents the signal undergoing the timing violation test. For example, the “D” signal of a D-flip-flop.
- TestPortName . . . .::= If the name of the signal is passed to this parameter as a STRING, then the signal name will appear in the assertion statement called by the subprogram.
- RefPort. . . . .::= A SIGNAL parameter which represents the reference signal against which the TestPort is tested. For example, the “CLK” signal of a D-flip-flop.
- RefPortName .. . .::= If the name of the signal is passed to this parameter as a STRING, then the signal name will appear in the assertion statement called by the subprogram.
- condition . . . .::= If TRUE, then the check will be performed; otherwise no check will be performed and no assertions reported. If condition is FALSE, then the return value is FALSE.

  This parameter has been provided so that procedures can be used concurrently while maintaining a mechanism for disabling the timing check under certain circumstances.
- HeaderMsg . .. .::= This string will be included in the assertion message. This string may contain any additional message desired to be produced whenever an assertion occurs. However, it is customary to pass the instance path name of the model which is testing the timing violation.

1.3 VITAL_TIMING Data Types

TransitionType is defined to provide from-to relationships between any of the states {0,1,Z} and any other state within that same set of three states.

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

DelayTypes are defined to provide with with two or six values of time.

```
SUBTYPE DelayTypeXX         is TIME;
SUBTYPE DelayType01         is TransitionArrayType (tr01 to tr10);
SUBTYPE DelayType01Z        is TransitionArrayType (tr01 to trz0);
```

Vectorized forms are also provided.

```
TYPE DelayArrayTypeXX       is ARRAY (natural range <>) of DelayTypeXX;
TYPE DelayArrayType01       is ARRAY (natural range <>) of DelayType01;
TYPE DelayArrayType01Z      is ARRAY (natural range <>) of DelayType01Z;
```

TimeArray is provided for event recording purposes.

```
TYPE TimeArray             is ARRAY (NATURAL RANGE <>) OF TIME;
```

VitalPropagateWireDelay . . . . handles wire delays

PURPOSE:

To delay an input or output port by the appropriate wire delay

DECLARATION:

```
PROCEDURE VitalPropagateWireDelay
    SIGNAL      OutSig      : OUT   std_logic;  -- signal delayed by port or net delay
    SIGNAL      InSig       : IN    std_logic;        -- input port
    CONSTANT    twire       : IN    DelayType01Z;     -- input port delay
    CONSTANT    t_hold_hi   : IN    TIME := 0 ns;     -- hold spec for (InSig='1')
    CONSTANT    t_hold_lo   : IN    TIME := 0 ns      -- hold spec for (InSig='0')

 ):
```

DESCRIPTION:

Performs the following function:

```
if ( (t_hold_hi < 0 ns) or (t_hold_lo < 0 ns) ) then
  delay := ABS(MINIMUM(t_hold_hi, t_hold_lo));
end if;
outsig <= TRANSPORT insig after
         (VitalCalcDelay( insig, insig'LAST_VALUE, twire ) + delay);
```

EXAMPLE:

```
SIGNAL clock_ipd : std_logic; -- internally wire delayed clock
SIGNAL databus_ipd : std_logic_vector (databus'range);

WIRE_DELAY : block
begin
    VitalPropagateWireDelay( outsig => clock_ipd,
                             insig  => clock,
                             twire  = ExtendToFillDelay(twire_clock));

    -- 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  = ExtendToFillDelay(twire_databus));
     end generate;
end block;
```

VitalTimingCheck .... Checks for Setup and Hold Violations

PURPOSE:

This procedure (a) detects the presence of a setup or hold violation on the "TestPort" signal with respect to the "RefPort", (b) sets a flag named "violation" accordingly and (c) issues an appropriate error message.

OVERLOADED DECLARATIONS:

Violation return type is Boolean:

```
PROCEDURE VitalTimingCheck (
    SIGNAL      TestPort          : IN     std_ulogic;       -- SIGNAL being tested
    CONSTANT    TestPortName      : IN     STRING := "";     -- name OF the signal
    SIGNAL      RefPort           : IN     std_ulogic;       -- SIGNAL being referenced
    CONSTANT    RefPortName       : IN     STRING := "";     -- name OF the ref signal
    CONSTANT    t_setup_hi        : IN     TIME := 0 ns;     -- setup for TestPort = '1'
    CONSTANT    t_setup_lo        : IN     TIME := 0 ns;     -- setup for TestPort = '0'
    CONSTANT    t_hold_hi         : IN     TIME := 0 ns;     -- hold for TestPort = '1'
    CONSTANT    t_hold_lo         : IN     TIME := 0 ns;     -- hold for TestPort = '0'
    CONSTANT    CheckEnabled      : IN     BOOLEAN;          -- true ==> spec checked.
    CONSTANT    RefTransition     : IN     BOOLEAN;          -- specify reference edge
                                           -- i.e. CLK = '1' for rising edge
    CONSTANT    HeaderMsg         : IN     STRING := " ";
    VARIABLE    TimeMarker        : INOUT  TimeMarkerType;
                                           -- holds time of last reference transition
                                           -- and the last time a hold check passed
    VARIABLE    Violation         : OUT    BOOLEAN);

PROCEDURE VitalTimingCheck (
    SIGNAL      TestPort          : IN     std_ulogic;       -- SIGNAL being tested
    CONSTANT    TestPortName      : IN     STRING := "";     -- name OF the signal
    SIGNAL      RefPort           : IN     std_ulogic;       -- SIGNAL being referenced
    CONSTANT    RefPortName       : IN     STRING := "";     -- name OF the ref signal
    CONSTANT    t_setup_hi        : IN     TIME := 0 ns;     -- setup for TestPort = '1'
    CONSTANT    t_setup_lo        : IN     TIME := 0 ns;     -- setup for TestPort = '0'
    CONSTANT    t_hold_hi         : IN     TIME := 0 ns;     -- hold for TestPort = '1'
    CONSTANT    t_hold_lo         : IN     TIME := 0 ns;     -- hold for TestPort = '0'
    CONSTANT    CheckEnabled      : IN     BOOLEAN;          -- true ==> spec checked.
    CONSTANT    RefTransition     : IN     BOOLEAN;          -- specify reference edge
                                           -- i.e. CLK = '1' for rising edge
    CONSTANT    HeaderMsg         : IN     STRING := " ";
    VARIABLE    TimeMarker        : INOUT  TimeMarkerType;
                                           -- holds time of last reference transition
                                           -- and the last time a hold check passed
    VARIABLE    TestPortLastEvent : INOUT  DelayArrayTypeXX;
                                           -- records time of test port events
    VARIABLE    TestPortLastValue : INOUT  std_logic_vector;
                                           -- records previous test port values
    VARIABLE    Violation         : OUT    BOOLEAN);
```

```
PROCEDURE VitalTimingCheck (
      SIGNAL        TestPort              : IN     std_ulogic;          -- SIGNAL being tested
      CONSTANT      TestPortName          : IN     STRING := "";        -- name OF the signal
      SIGNAL        RefPort               : IN     std_ulogic;          -- SIGNAL being referenced
      CONSTANT      RefPortName           : IN     STRING := "";        -- name OF the ref signal
      CONSTANT      t_setup_hi            : IN     TIME := 0 ns;        -- setup for TestPort = '1'
      CONSTANT      t_setup_lo            : IN     TIME := 0 ns;        -- setup for TestPort = '0'
      CONSTANT      t_hold_hi             : IN     TIME := 0 ns;        -- hold for TestPort = '1'
      CONSTANT      t_hold_lo             : IN     TIME := 0 ns;        -- hold for TestPort = '0'
      CONSTANT      CheckEnabled          : IN     BOOLEAN;             -- true ==> spec checked.
      CONSTANT      RefTransition         : IN     BOOLEAN;             -- specify reference edge
                                                   -- i.e. CLK = '1' for rising edge
      CONSTANT      HeaderMsg             : IN     STRING := " ";
      VARIABLE      TimeMarker            : INOUT  TimeMarkerType;
                                                   -- holds time of last reference transition
                                                   -- and the last time a hold check passed
      VARIABLE      TestPortLastEvent     : INOUT  DelayArrayTypeXX;
                                                   -- records time of test port events
      VARIABLE      TestPortLastValue     : INOUT  std_ulogic_vector;
                                                   -- records previous test port values
      VARIABLE      Violation             : OUT    BOOLEAN);
```

Violation return type is X01:

'X' indicates that a violation HAS occured.
(HINT: you may use this value along with an XOR gate for the purpose of Xgeneration)

'0' indicates that a violation HAS NOT occured.

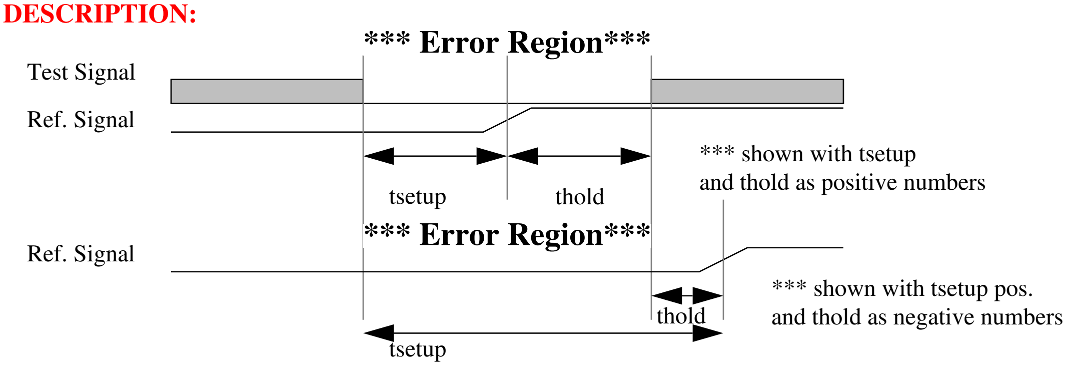
'1' is never returned.

```
PROCEDURE VitalTimingCheck (
      SIGNAL        TestPort              : IN     std_ulogic;          -- SIGNAL being tested
      CONSTANT      TestPortName          : IN     STRING := "";        -- name OF the signal
      SIGNAL        RefPort               : IN     std_ulogic;          -- SIGNAL being referenced
      CONSTANT      RefPortName           : IN     STRING := "";        -- name OF the ref signal
      CONSTANT      t_setup_hi            : IN     TIME := 0 ns;        -- setup for TestPort = '1'
      CONSTANT      t_setup_lo            : IN     TIME := 0 ns;        -- setup for TestPort = '0'
      CONSTANT      t_hold_hi             : IN     TIME := 0 ns;        -- hold for TestPort = '1'
      CONSTANT      t_hold_lo             : IN     TIME := 0 ns;        -- hold for TestPort = '0'
      CONSTANT      CheckEnabled          : IN     BOOLEAN;             -- true ==> spec checked.
      CONSTANT      RefTransition         : IN     BOOLEAN;             -- specify reference edge
                                                   -- i.e. CLK = '1' for rising edge
      CONSTANT      HeaderMsg             : IN     STRING := " ";
      VARIABLE      TimeMarker            : INOUT  TimeMarkerType;
                                                   -- holds time of last reference transition
                                                   -- and the last time a hold check passed
      VARIABLE      Violation             : OUT    X01);
```

```
PROCEDURE VitalTimingCheck (
    SIGNAL      TestPort            : IN     std_ulogic;       -- SIGNAL being tested
    CONSTANT    TestPortName        : IN     STRING := "";     -- name OF the signal
    SIGNAL      RefPort             : IN     std_ulogic;       -- SIGNAL being referenced
    CONSTANT    RefPortName         : IN     STRING := "";     -- name OF the ref signal
    CONSTANT    t_setup_hi          : IN     TIME := 0 ns;     -- setup for TestPort = '1'
    CONSTANT    t_setup_lo          : IN     TIME := 0 ns;     -- setup for TestPort = '0'
    CONSTANT    t_hold_hi           : IN     TIME := 0 ns;     -- hold for TestPort = '1'
    CONSTANT    t_hold_lo           : IN     TIME := 0 ns;     -- hold for TestPort = '0'
    CONSTANT    CheckEnabled        : IN     BOOLEAN;          -- true ==> spec checked.
    CONSTANT    RefTransition       : IN     BOOLEAN;          -- specify reference edge
                                             -- i.e. CLK = '1' for rising edge
    CONSTANT    HeaderMsg           : IN     STRING := " ";
    VARIABLE    TimeMarker          : INOUT  TimeMarkerType;
                                             -- holds time of last reference transition
                                             -- and the last time a hold check passed
    VARIABLE    TestPortLastEvent   : INOUT  DelayArrayTypeXX;
                                             -- records time of test port events
    VARIABLE    TestPortLastValue   : INOUT  std_logic_vector;
                                             -- records previous test port values
    VARIABLE    Violation           : OUT    X01);

PROCEDURE VitalTimingCheck (
    SIGNAL      TestPort            : IN     std_ulogic;       -- SIGNAL being tested
    CONSTANT    TestPortName        : IN     STRING := "";     -- name OF the signal
    SIGNAL      RefPort             : IN     std_ulogic;       -- SIGNAL being referenced
    CONSTANT    RefPortName         : IN     STRING := "";     -- name OF the ref signal
    CONSTANT    t_setup_hi          : IN     TIME := 0 ns;     -- setup for TestPort = '1'
    CONSTANT    t_setup_lo          : IN     TIME := 0 ns;     -- setup for TestPort = '0'
    CONSTANT    t_hold_hi           : IN     TIME := 0 ns;     -- hold for TestPort = '1'
    CONSTANT    t_hold_lo           : IN     TIME := 0 ns;     -- hold for TestPort = '0'
    CONSTANT    CheckEnabled        : IN     BOOLEAN;          -- true ==> spec checked.
    CONSTANT    RefTransition       : IN     BOOLEAN;          -- specify reference edge
                                             -- i.e. CLK = '1' for rising edge
    CONSTANT    HeaderMsg           : IN     STRING := " ";
    VARIABLE    TimeMarker          : INOUT  TimeMarkerType;
                                             -- holds time of last reference transition
                                             -- and the last time a hold check passed
    VARIABLE    TestPortLastEvent   : INOUT  DelayArrayTypeXX;
                                             -- records time of test port events
    VARIABLE    TestPortLastValue   : INOUT  std_ulogic_vector;
                                             -- records previous test port values
    VARIABLE    Violation           : OUT    X01);
```

DESCRIPTION:



- t_setup_hi ::= . . . . .Absolute minimum time duration before the transition of the RefPort for which transitions of TestPort are allowed to proceed to the to_X01(TestPort) ='1' state without causing a setup violation.
- t_setup_lo ::= . . . . .Absolute minimum time duration before the transition of the RefPort for which transitions of TestPort are allowed to proceed to the to_X01(TestPort) ='0' state without causing a setup violation.
- t_hold_hi ::= . . . . .Absolute minimum time duration after the transition of the RefPort for which transitions of TestPort are allowed to proceed to the to_X01(TestPort) ='1' state without causing a hold violation.
- t_hold_lo ::= . . . . .Absolute minimum time duration after the transition of the RefPort for which transitions of TestPort are allowed to proceed to the to_X01(TestPort) ='0' state without causing a hold violation.
- CheckEnabled ::= . . . . .The function immediately checks CheckEnabled for a TRUE value. If the value is FALSE, the FUNCTION immediately returns with the value of FALSE (‘0’ if the Violation type is X01).
- RefTransition ::=. . . . . .This specifies the condition on which an event on the RefPort will be considered to be the reference event. For instance (CLK=’1’) means that the reference event is any transition to ‘1’ by the signal CLK.
- HeaderMsg ::= . . . . .This HeaderMsg string will accompany any assertion messages produced by this function.
- TimeMarker ::= . . . . .Holds the time of the last reference transition and the last time that a hold check passed. Must be associated with an aggregate which initializes this variable. TimeMarker => (-500 ns, NULL, NULL) The user should never change the state of that variable
- Violation ::= .. . . .BOOLEAN: This routine will set the flag TRUE if a violation exists and will set the flag FALSE if no violation exists.
  X01: The routine will produce an ‘X’ if a violation occured, ‘0’ otherwise.

For vectors:

- TestPortLastEvent ::= . . . . .This VARIABLE is used to store the last time in which TestPort changed value.
- TestPortLastValue ::= . . . . .This VARIABLE is used to store the last value taken on by TestPort prior to its current value.

ASSUMPTIONS:

a. For positive hold times both a setup and a hold error may be reported.

b. For both hold times negative, only one error may be reported per clock cycle and that error will be a setup error.

c. For one negative hold time the procedure will only report one error (setup or hold) per clock cycle.

d. Negative setup times are not allowed.

e. Regardless of whether CheckEnabled is true, the transitions of the TestPort are recorded by the routine. In order for this routine to work correctly, this procedure shall not be placed in a branch other than to disable all timing checks.

BUILT IN ERROR TRAPS:

If the observed setup or hold time is less than the specified values, an assertion message will be issued and the variable “Violation” will be set TRUE. Otherwise, no message will be asserted and the variable “Violation” will be set FALSE.

EXAMPLE:

```
VARIBALE TimeMarkerRLCLK : TimeMarkerType := (-500 ns, NULL, NULL);
--------------------
-- Timing Check Section
--------------------
IF (TimingCheckON) THEN
  -- setup RL high or low before rising CLK
  --  hold RL high or LOW after  rising CLK
  VitalTimingCheck (
         TestPort      =>     RL_CLK_ipd,
         TestPortName  =>     "RL",
         RefPort       =>     CLK_ipd,
         RefPortName   =>     "CLK",
         t_setup_hi    =>     tsetup_RL_CLK(tr01),
         t_setup_lo    =>     tsetup_RL_CLK(tr10),
         t_hold_hi     =>     thold_RL_CLK(tr01),
         t_hold_lo     =>     thold_RL_CLK(tr10),
         CheckEnabled  =>     TRUE,
         RefTRansition =>     (CLK_ipd='1'),
         HeaderMsg     =>     InstancePath & "/DLE24",
         TimeMarker    =>     TimeMarkerRLCLK,
         Violation     =>     Tviol_RL_CLK );

END IF; -- Timing Check Section
```

VitalSetupHoldCheck . . . . Check Setup & Hold times

PURPOSE:

This procedure (a) detects the presence of a setup or hold violation on the "TestPort" signal with respect to the "RefPort", (b) sets a flag named "violation" accordingly and (c) issues an appropriate error message.

This procedure differs from the VitalTimingCheck procedure in that this procedure does not issue any assertion messages in the event of a timing violation, whereas the VitalTimingCheck procedure issues its own error messages. In order to issue error messages, this procedure must be used in conjunction with the VitalReportSetupHoldViolation and/or VitalReportRlseRmvlViolation procedures.

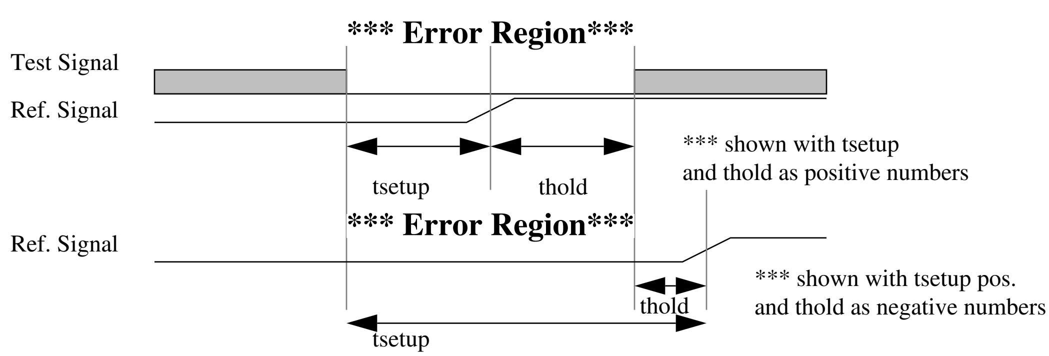
DECLARATION:

```
PROCEDURE VitalSetupHoldCheck (
    SIGNAL      TestPort        : IN     std_ulogic;     -- SIGNAL being tested
    SIGNAL      RefPort         : IN     std_ulogic;     -- SIGNAL being referenced
    CONSTANT    t_setup_hi      : IN     TIME := 0 ns;   -- setup for TestPort = '1'
    CONSTANT    t_setup_lo      : IN     TIME := 0 ns;   -- setup for TestPort = '0'
    CONSTANT    t_hold_hi       : IN     TIME := 0 ns;   -- hold for TestPort = '1'
    CONSTANT    t_hold_lo       : IN     TIME := 0 ns;   -- hold for TestPort = '0'
    CONSTANT    CheckEnabled    : IN     BOOLEAN;        -- true ==> spec checked.
    CONSTANT    RefTransition   : IN     BOOLEAN;        -- specify reference edge
                                         -- i.e. CLK = '1' for rising edge
    VARIABLE    TimeMarker      : INOUT  TimeMarkerType;
                                         -- holds time of last reference transition
                                         -- and the last time a hold check passed
    VARIABLE    TimingInfo      : OUT    TimingInfoType  -- violation information
);
```

APPLICABLE TYPES:

```
TYPE ViolationType is ( NoViolation, SetupViolation, HoldViolation );
TYPE TimingInfoType is RECORD
        Violation    : ViolationType;     -- kind of violation which occured
        ObservedTime : time;              -- when the violation occured
        ExpectedTime : time;              -- when the transition was expected
        ConstrntTime : time;              -- what the spec stated
        State        : X01;               -- state of the tested signal
END RECORD;
```

DESCRIPTION:



- t_setup_hi ::= . . . . .Absolute minimum time duration before the transition of the RefPort for which transitions of TestPort are allowed to proceed to the '1' state without causing a setup violation.
- t_setup_lo ::= . . . . .Absolute minimum time duration before the transition of the RefPort for which transitions of TestPort are allowed to proceed to the '0' state without causing a setup violation.
- t_hold_hi ::= . . . . .Absolute minimum time duration after the transition of the RefPort for which transitions of TestPort are allowed to proceed to the '1' state without causing a hold violation.
- t_hold_lo ::= . . . . .Absolute minimum time duration after the transition of the RefPort for which transitions of TestPort are allowed to proceed to the '0' state without causing a hold violation.
- CheckEnabled ::= . . . . .The function immediately checks CheckEnabled for a TRUE value. If the value is FALSE, the FUNCTION immediately returns with the value of FALSE (‘X’ if the Violation type is X01).
- RefTransition ::=. . . . . .This specifies the condition on which an event on the RefPort will be considered to be the reference event. For instance (CLK=’1’) means that the reference event is any transition to ‘1’ by the signal CLK.
- TimeMarker ::= . . . . .Holds the time of the last reference transition and the last time that a hold check passed. Must be associated with an aggregate which initializes this variable. TimeMarker => (-500 ns, NULL, NULL) The user should never change the state of that variable
- TimingInfo ::= . . . . .This the violation information

ASSUMPTIONS:

a. For positive hold times both a setup and a hold error may be reported.

b. For both hold times negative, only one error may be reported per clock cycle and that error will be a setup error.

c. For one negative hold time the procedure will only report one error (setup or hold) per clock cycle.

d. Negative setup times are not allowed.

e. Regardless of whether CheckEnabled is true, the transitions of the TestPort are recorded by the routine. In order for this routine to work correctly, this procedure shall not be placed in a branch other than to disable all timing checks.

EXAMPLE:

```
VARIABLE TimingInfoRLCLK : TImingInfoType;
VARIBALE TimeMarketRLCLK : TimeMarkerType := (-500 ns, NULL, NULL);
--------------------
-- Timing Check Section
--------------------
IF (TimingCheckON) THEN
  -- setup RL high or low before rising CLK
  --  hold RL high or LOW after  rising CLK
  VitalTimingCheck (
        TestPort      =>     RL_CLK_ipd,
        RefPort       =>     CLK_ipd,
        t_setup_hi    =>     tsetup_RL_CLK(tr01),
        t_setup_lo    =>     tsetup_RL_CLK(tr10),
        t_hold_hi     =>     thold_RL_CLK(tr01),
        t_hold_lo     =>     thold_RL_CLK(tr10),
        CheckEnabled  =>     TRUE,
        RefTRansition =>     (CLK_ipd='1'),
        TimeMarker    =>     TimeMarkerRLCLK,
        TimingInfo    =>     TimingInfoRLCLK );
  VitalReportSetupHoldViolation (
        TestPortName  =>     "RL",
        RefPortName   =>     "CLK",
        HeaderMsg     =>     InstancePath & "/DLE24",
        TimingInfo    =>     TimingInfoRLCLK);
  Violation :=  TimingInfoRLCLK.Violation /= NoViolation;

END IF; -- Timing Check Section
```

VitalReportSetupHoldViolation

PURPOSE:

This routine issues an ERROR level assertion which includes: Header Message, Violation Type (i.e. Setup, Hold), State of the Violation (i.e. HI, LO), the name of the Tested Signal, the name of the Reference Signal upon which the test was based, the Expected time response, the Observed time response and the time at which the violation occurred.

DECLARATION:

```
PROCEDURE VitalReportSetupHoldViolation (
    CONSTANT    TestPortName:    IN string := "";    -- name of the tested signal
    CONSTANT    RefPortName:     IN string := "";    -- name of the reference signal
    CONSTANT    HeaderMsg:       IN string := " ";
    CONSTANT    TimingInfo:      IN TimingInfoType   -- Timing Violation Information
);
```

DESCRIPTION:

Issues NO assertion message unless TimingInfo.Violation /= NoViolation. Otherwise, this procedure will issue ERROR level assertion messages reporting the details of the failed setup/hold violation.

- TestPortName ::= . . . . .STRING name of the port undergoing the violation test.
- RefPortName ::= . . . . .STRING name of the port against which the testport's transitions are referenced.
- HeaderMsg ::= . . . . .This HeaderMsg string will accompany any assertion messages produced by this function.
- TimingInfo ::= . . . . .This variable RECORD holds information from a previous usage of VitalSetupHoldCheck

APPLICABLE TYPES:

```
TYPE ViolationType is ( NoViolation, SetupViolation, HoldViolation );
TYPE TimingInfoType is RECORD
        Violation     : ViolationType;    -- kind of violation which occured
        ObservedTime  : time;             -- when the violation occured
        ExpectedTime  : time;             -- when the transition was expected
        ConstrntTime  : time;             -- what the spec stated
        State         : X01;              -- state of the tested signal
END RECORD;
```

VitalReportRlseRmvlViolation

PURPOSE:

This routine issues an ERROR level assertion which includes: Header Message, Violation Type (i.e. Release, Removal), State of the Violation (i.e. HI, LO), the name of the Tested Signal, the name of the Reference Signal upon which the test was based, the Expected time response, the Observed time response and the time at which the violation occurred.

DECLARATION:

```
PROCEDURE VitalReportRlseRmvViolation (
    CONSTANT    TestPortName:     IN string := "";    -- name of the tested signal
    CONSTANT    RefPortName:      IN string := "";    -- name of the reference signal
    CONSTANT    HeaderMsg:        IN string := " ";
    CONSTANT    TimingInfo:       IN TimingInfoType   -- Timing Violation Information
);
```

DESCRIPTION:

Issues NO assertion message unless TimingInfo.Violation /= NoViolation. Otherwise, this procedure will issue ERROR level assertion messages reporting the details of the failed release/removal violation.

- TestPortName ::= . . . . .STRING name of the port undergoing the violation test.
- RefPortName ::= . . . . .STRING name of the port against which the testport's transitions are referenced.
- HeaderMsg ::= . . . . .This HeaderMsg string will accompany any assertion messages produced by this function.
- TimingInfo ::= . . . . .This variable RECORD holds information from a previous usage of VitalSetupHoldCheck

APPLICABLE TYPES:

```
TYPE ViolationType is ( NoViolation, SetupViolation, HoldViolation );
TYPE TimingInfoType is RECORD
        Violation     : ViolationType;     -- kind of violation which occured
        ObservedTime  : time;              -- when the violation occured
        ExpectedTime  : time;              -- when the transition was expected
        ConstrntTime  : time;              -- what the spec stated
        State         : X01;               -- state of the tested signal
END RECORD;
```

VitalPeriodCheck .... Checks for Periodicity Violations

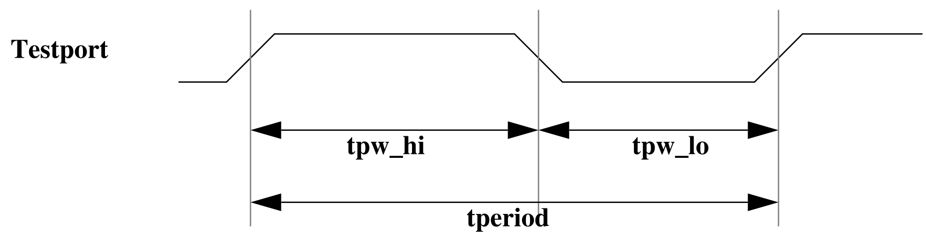
PURPOSE:

This procedure checks for minimum and maximum periodicity and pulse

DECLARATION:

```
PROCEDURE VItalPeriodCheck (
    SIGNAL      TestPort  :       IN std_ulogic;
    CONSTANT    TestPortName  :   IN STRING := “”;
    CONSTANT    PeriodMin  :      IN TIME := 0 ns;
    CONSTANT    PeriodMax  :      IN TIME := TIME’HIGH;
    CONSTANT    pw_hi_min  :      IN TIME := 0 ns;
    CONSTANT    pw_hi_max:        IN TIME := TIME’HIGH;
    CONSTANT    pw_lo_min  :      IN TIME := 0 ns;
    CONSTANT    pw_lo_max  :      IN TIME := TIME’HIGH;
    VARIABLE    info      :       INOUT DelayArrayTypeXX :=  PeriodCheckInfo_Init;
    VARIABLE    Violation  :      OUT BOOLEAN;
    CONSTANT    HeaderMsg  :      IN STRING := “”;
    CONSTANT    Condition:        IN Boolean
    );
```

DESCRIPTION:



This procedure must be used in conjunction with a process statement in order to retain the transition times to '0' and '1'

- PeriodMin ::= ....Minimum allowable time period between successive rising or falling edges of the TestPort.
- PeriodMax ::=............Maximum allowable time period between successive rising or falling edges of the TestPort.
- pw_hi_min ::= ....Minimum allowable time period during which Testport is maintained at a ‘1’ or ‘H’ value.
- pw_hi_max ::= ....Maximum allowable time period during which Testport is maintained at a ‘1’ or ‘H’ value.
- pw_lo_min ::= ....Minimum allowable time period during which Testport is maintained at a ‘0’ or ‘L’ value.
- pw_lo_max ::= ....Maximum allowable time period during which Testport is maintained at a ‘0’ or ‘L’ value.
- info ::=....... ....VARIABLE parameter which records the transition times for use internal to the routine.
- Violation ::=. .. ..TRUE if either the pulsewidth or period has been violated.
- HeaderMsg ::= ....STRING of information commonly used to indicate the clling Instance.

BUILT IN ERROR TRAPS:

Assertion messages are generated if any of the timing specifications are violated.

EXAMPLE:

```
--------------------
-- Timing Check Section
--------------------
IF (TimingCheckON) THEN
  -- Pulse width, period check CLK
  VitalPeriodCheck (
         testport          =>CLK_ipd,
         testportname      =>   "CLK_ipd",
         periodmin         =>   tperiod_CLK,
         periodmax         =>   TIME'HIGH,
         pw_hi_min         =>   tpw_CLK(tr01),
         pw_hi_max         =>   TIME'HIGH,
         pw_lo_min         =>   tpw_CLK(tr10),
         pw_lo_max         =>   TIME'HIGH,
         info              =>   PeriodCheckInfo_CLK,
         Violation         =>   Pviol_CLK,
         HeaderMsg         =>   InstancePath & "/DLE24",
        Condition         => TRUE
        );

END IF; -- Timing Check Section
```

VitalExtendToFillDelay

. . . Extends delays to 6-element format

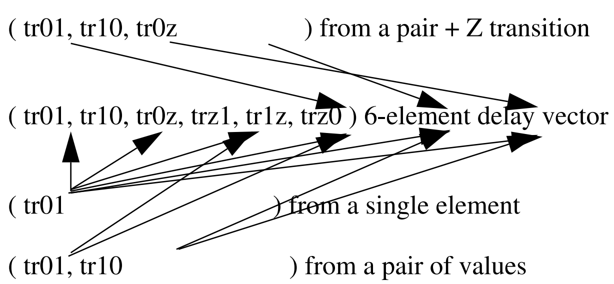
PURPOSE:

To provide a set of six transition dependent time values for use in delay assignments even though only 1,2 or 3 values of delay may have been provided.

DECLARATION:

```
FUNCTION VitalExtendToFillDelay (
    CONSTANT     delay   : IN            TransitionArrayType
) RETURN DelayType01Z;
```

DESCRIPTION:



ALGORITHM:

```
VARIABLE d_val  : DelayType01Z;
BEGIN
  CASE delay'length IS
        WHEN  1  =>  d_val := (others => delay(tr01));
        WHEN  2  =>  d_val(tr01) := delay(tr01);
                     d_val(tr0z) := delay(tr01);
                     d_val(trz1) := delay(tr01);
                     d_val(tr10) := delay(tr10);
                     d_val(tr1z) := delay(tr10);
                     d_val(trz0) := delay(tr10);
        WHEN  3  =>  d_val(tr01) := delay(tr01);
                     d_val(trz1) := delay(tr01);
                     d_val(tr10) := delay(tr10);
                     d_val(trz0) := delay(tr10);
                     d_val(tr0z) := delay(tr0z);
                     d_val(tr1z) := delay(tr0z);
        WHEN  6  =>  d_val := delay;
        WHEN  others  => assert false
              report "VitalExtendToFillDelay(delay'length /= [1,2,3,6])"
              SEVERITY ERROR;
  END CASE;
RETURN (d_val);
```

EXAMPLE:

Variable Verilog_Delay_Vector: DelayType01Z;
Verilog_Delay_Vector := VitalExtendToFillDelay ((1 ns, 2 ns));

VitalCalcDelay ...... Calculate Output Delays

PURPOSE:

To calculate the output delays based upon the signal values.

OVERLOADED DECLARATIONS:

```
FUNCTION VitalCalcDelay (
      CONSTANT    newval  :     IN std_ulogic;  -- new value of signal
      CONSTANT    oldval  :     IN std_ulogic;  -- old value of signal
      CONSTANT    delay   :     IN TranitionArrayType := UnitDelay
) RETURN DelayTypeXX;

FUNCTION VitalCalcDelay (
      CONSTANT    value   :     IN std_ulogic;  -- new value of signal
      CONSTANT    delay   :     IN TranitionArrayType := UnitDelay
) RETURN DelayTypeXX;
```

DESCRIPTION:

This function determines the proper value of delay to use given the newly assigned value and the existing value on the signal or driver. For the two parameter versions, only the newly assigned value is used in determining the delay.

EXAMPLES:

In this example, the old signal value is not considered. Hence the function will determine which delay to select based upon only the new value.

```
y <= the_new_value AFTER
  VitalCalcDelay(
        newval => the_new_value,
        oldval => '-',
        delay  => (1 ns, 2 ns, 3 ns, 4 ns, 5 ns, 6 ns)
  );
```

ALGORITHM:

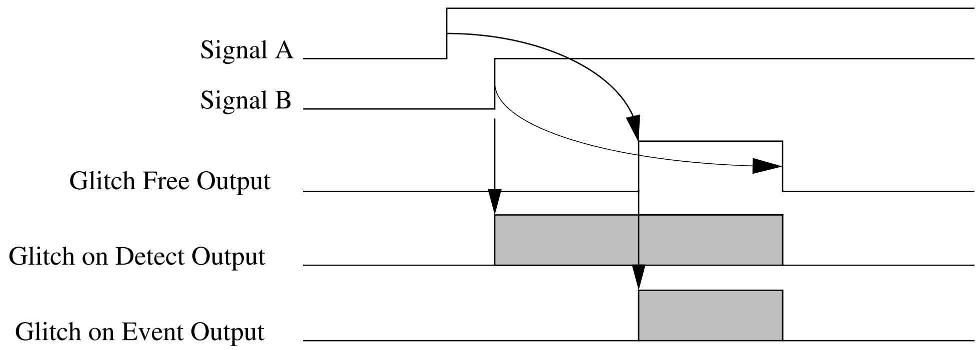
```
if (delay'LENGTH /= 6) then
    d_val := VitalExtendToFillDelay(delay);
else
    d_val := delay;
end if;
CASE oldval IS
  WHEN '0' |
       'L'  =>  CASE newval IS
                    WHEN '0' |
                         'L'    => result := d_val(tr10);
                    WHEN '1' |
                         'H'    => result := d_val(tr01);
                    WHEN 'Z'    => result := d_val(tr0z);
                    WHEN OTHERS => result := MINIMUM(d_val(tr01), d_val(tr0z));
                END CASE;
  WHEN '1' |
       'H'  =>  CASE newval IS
                    WHEN '0' |
                         'L'    => result := d_val(tr10);
                    WHEN '1' |
                         'H'    => result := d_val(tr01);
                    WHEN 'Z'    => result := d_val(tr1z);
                    WHEN OTHERS => result := MINIMUM(d_val(tr10), d_val(tr1z));
                END CASE;
  WHEN 'Z'  =>  CASE newval IS
                    WHEN '0' |
                         'L'    => result := d_val(trz0);
                    WHEN '1' |
                         'H'    => result := d_val(trz1);
                    WHEN 'Z'    => result := MAXIMUM (d_val(tr0z), d_val(tr1z));
                    WHEN OTHERS => result := MINIMUM (d_val(trz1), d_val(trz0));
                END CASE;
  WHEN 'U' |
       'X' |
       'W' |
       '-'  =>  CASE newval IS
                    WHEN '0' |
                         'L'    => result := MAXIMUM(d_val(tr10), d_val(trz0));
                    WHEN '1' |
                         'H'    => result := MAXIMUM(d_val(tr01), d_val(trz1));
                    WHEN 'Z'    => result := MAXIMUM(d_val(tr1z), d_val(tr0z));
                    WHEN OTHERS => result := MAXIMUM(d_val(tr10), d_val(tr01));
                END CASE;
  END CASE;
  RETURN result;
```

VitalGlitchOnEvent . . .. Handles Glitch Propagation on Events

PURPOSE:

A GLITCH is said to occur whenever a new assignment is scheduled to occur at an absolute time which is more than the absolute time of a previously scheduled pending event.

DECLARATION:




```
PROCEDURE VitalGlitchOnEvent (
      SIGNAL      OutSignal :        OUT    std_logic;          -- signal being driven
      CONSTANT    OutSignalName:     IN     string;             -- name of the signal
      VARIABLE    GlitchData :       INOUT  GlitchDataType;     -- Internal glitch data
      CONSTANT    NewValue:          IN     std_logic;          -- value being assigned
      CONSTANT    NewDelay:          IN     TIME  := 0 ns;      -- delay value
      CONSTANT    GlitchMode:        IN     GlitchModeType      := MessagePlusX;
      CONSTANT    GlitchDelay:       IN     TIME  := 0 ns       -- delay to possible glitch
);
PROCEDURE VitalGlitchOnEvent (
      SIGNAL      OutSignal :        OUT    std_logic_vector;      -- signal being driven
      CONSTANT    OutSignalName:     IN     string;                -- name of the signal
      VARIABLE    GlitchData :       INOUT  GlitchDataArrayType;      -- Internal glitch data
      CONSTANT    NewValue:          IN     std_logic_vector;      -- value being assigned
      CONSTANT    NewDelay:          IN     TimeArray;             -- delay value
      CONSTANT    GlitchMode:        IN     GlitchModeType         := MessagePlusX;
      CONSTANT    GlitchDelay:       IN     TIMEArray  := 0 ns       -- delay to possible glitch
);
```

DESCRIPTION:

- OutSignal. . . .. .Signal being driven
- OutSignalName . . . . .Name of the driven signal
- GlitchData . . .. .Internal data required by the procedure
- NewValue . . .. .new value being assigned
- NewDelay . . .. .Delay accompanying the assignment (Note: for vectors, this is an array)
- GlitchMode . .. . .MessagePlusX,. . .. . X generation ON, Messaging ON
  MessageOnly,. . . X generation OFF, Messaging OFF
  XOnly, ... X generation ON, Messaging OFF
  NoGlitch . . . . X generation OFF, Messaging OFF

- GlitchDelay . .. . .if <= 0 ns , then there will be no Glitch
  if > NewDelay, then there is no Glitch, otherwise, this is the relative time from NOW when a FORCED generation of a glitch will occur.

ASSUMPTIONS:

none.

BUILT IN ERROR TRAPS:

If GlitchMode is set to MessagePlusX or MessageOnly, then an assertion will be issued whenever a glitch is detected. The assertion will be made exactly at the time in which the glitch is detected, regardless of whether GlitchOnEvent or GlitchOnDetect was used.

```
APPLICABLE TYPES:
TYPE GlitchModeType is ( MessagePlusX, MessageOnly, XOnly, NoGlitch );

TYPE GlitchDataType IS
    RECORD
        LastSchedTransaction  : TIME;
        LastGlitchSchedTime   : TIME;
        LastSchedValue        : std_ulogic;
        CurrentValue          : std_ulogic;
    END RECORD;

TYPE GlitchDataArrayType IS ARRAY (natural range <>) of GlitchDataType;
```

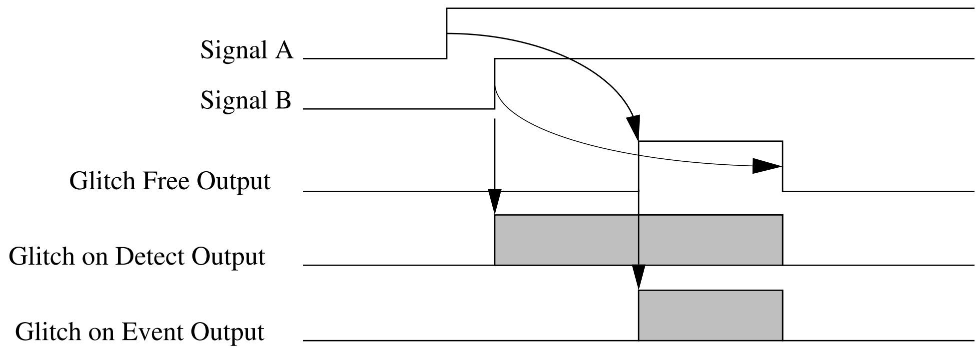
VitalGlitchOnDetect

Handles Glitch Propagation on Detection

PURPOSE:

A GLITCH is said to occur whenever a new assignment is scheduled to occur at an absolute time which is more than the absolute time of a previously scheduled pending event.

DECLARATION:




```
PROCEDURE VitalGlitchOnDetect (
    SIGNAL      OutSignal :       OUT    std_logic;          -- signal being driven
    CONSTANT    OutSignalName:    IN     string;             -- name of the signal
    VARIABLE    GlitchData :      INOUT  GlitchDataType;     -- Internal glitch data
    CONSTANT    NewValue:         IN     std_logic;          -- value being assigned
    CONSTANT    NewDelay:         IN     TIME  := 0 ns;      -- delay value
    CONSTANT    GlitchMode:       IN     GlitchModeType      := MessagePlusX;
    CONSTANT    GlitchDelay:      IN     TIME  := 0 ns       -- delay to possible glitch
);
PROCEDURE VitalGlitchOnDetect (
    SIGNAL      OutSignal :       OUT    std_logic_vector;     -- signal being driven
    CONSTANT    OutSignalName:    IN     string;               -- name of the signal
    VARIABLE    GlitchData :      INOUT  GlitchDataArrayType;  -- Internal glitch data
    CONSTANT    NewValue:         IN     std_logic_vector;     -- value being assigned
    CONSTANT    NewDelay:         IN     TimeArray ;           -- delay value
    CONSTANT    GlitchMode:       IN     GlitchModeType        := MessagePlusX;
    CONSTANT    GlitchDelay:      IN     TIMEArray  := 0 ns    -- delay to possible glitch
);
```

DESCRIPTION:

- OutSignal. . . .. .Signal being driven
- OutSignalName . . . . .Name of the driven signal
- GlitchData . . .. .Internal data required by the procedure
- NewValue . . .. .new value being assigned
- NewDelay . . .. .Delay accompanying the assignment (Note: for vectors, this is an array)
- GlitchMode . .. . .MessagePlusX,. . .. . X generation ON, Messaging ON
  MessageOnly,. . . X generation OFF, Messaging OFF
  XOnly, ... X generation ON, Messaging OFF
  NoGlitch . . . . X generation OFF, Messaging OFF

- GlitchDelay . .. . .if <= 0 ns , then there will be no Glitch
  if > NewDelay, then there is no Glitch, otherwise, this is the relative time from NOW when a FORCED generation of a glitch will occur.

ASSUMPTIONS:

None.

BUILT IN ERROR TRAPS:

If GlitchMode is set to MessagePlusX or MessageOnly, then an assertion will be issued whenever a glitch is detected. The assertion will be made exactly at the time in which the glitch is detected, regardless of whether GlitchOnEvent or GlitchOnDetect was used..

```
APPLICABLE TYPES:
TYPE GlitchModeType is ( MessagePlusX, MessageOnly, XOnly, NoGlitch );

TYPE GlitchDataType IS
    RECORD
        LastSchedTransaction  : TIME;
        LastGlitchSchedTime   : TIME;
        LastSchedValue        : std_ulogic;
        CurrentValue          : std_ulogic;
    END RECORD;

TYPE GlitchDataArrayType IS ARRAY (natural range <>) of GlitchDataType;
```

EXAMPLES:

See Example on GlitchOnEvent.

VitalPropagatePathDelay . . . . Handles Pin-to-pin delay

PURPOSE:

To assign the correct delay to an output signal based upon changes which may have occurred on affecting input signals.

OVERLOADED DECLARATIONS:

```
PROCEDURE VitalPropagatePathDelay (
    SIGNAL      OutSignal        : OUT std_logic; -- output signal
    CONSTANT    OutSignalName    : IN    string; -- name of the output signal
    CONSTANT    OutTemp          : IN    std_logic;    -- intermediate 0-delay output
    CONSTANT    Paths            : IN    PathArrayType; -- all possible paths
    VARIABLE    GlitchData       : INOUT Glitchdatatype;
    CONSTANT    GlitchMode       : IN    Glitchmodetype ;
    CONSTANT    GlitchKind       : IN    Glitchkind := OnEvent -- glitch kind to be used
);
```

DESCRIPTION:

This procedure is to be used for accurate modeling of Path delays (conditional paths handling). It will select the appropriate path delay, based on PathCondition and schedule the new output value with the selected delay specified along with Glitch handling (OnEvent or OnDetect).

BUILT IN ERROR TRAPS:

None.

RELATED TYPES:

```
TYPE PathType IS RECORD
    InputChangeTime : time;         -- timestamp for path input signal
    PathDelay       : DelayType01Z;-- delay for this path
    PathCondition   : boolean;      -- condition which sensitizes this path
END RECORD;

TYPE PathArrayType is array (natural range <> ) of PathType;
TYPE GlitchKind is (OnEvent, On_Detect);
```

EXAMPLE:

```
VitalPropagatePathDelay(
    OutSignal     => QN,        -- Actual OUTPUT signal
    OutSignalName => "QN",
    OutTemp       => QN_zd,     -- Zero delayed internal Output signal
    -- First of four input pins which affect the output
    Paths(0).InputChangeTime => CLK_ipd'LAST_EVENT,
    Paths(0).PathDelay       => tpd_CLK_QN,
    Paths(0).Condition       => TRUE,
    -- Second of four input pins which affect the output
    Paths(1).InputChangeTime => PN_ipd'last_event,
    Paths(1).PathDelay       => tpd_PN_QN,
    Paths(1).Condition       => TRUE,
    -- Third of four input pins which affect the output
    Paths(2).InputChangeTime => CN_ipd'LAST_EVENT,
    Paths(2).PathDelay       => tpd_CN_QN,
    Paths(2).Condition       => TRUE,
    -- Fourth of four input pins which affect the output
    Paths(3).InputChangeTime => 0 ns,
    Paths(3).PathDelay       => tpd_zero,
    Paths(3).Condition       => ViolationFlag = '1'),
    GlitchData    => GlitchData,
    GlitchMode    => Xonly,
    GlitchKind    => OnEvent);
```

ALGORITHM:

```
variable Tlast, Tpath, T1:  TIME := TIME'RIGHT;
BEGIN
        FOR i IN Paths'RANGE LOOP
            if Paths(i).PathCondition = TRUE then
                if Paths(i).InputChangeTime < Tlast then
                    -- Get the delay from the most recent input change
                    Tpath := VitalCalcDelay(OutTemp, GlitchData.LastSchedValue, Paths(i).PathDelay);
                    Tlast  := Paths(i).InputChangeTime;
                elsif Paths(i).InputChangeTime = Tlast then
                    -- Simultaneous inputs change
                    T1 := VitalCalcDelay(OutTemp, GlitchData.LastSchedValue, Paths(i).PathDelay);
                   if T1 <= Tpath then -- This is the shortest path
                        Tpath := T1;
                   end if;
                end if;
            end if;
        END LOOP;
        -- Ensure that the correct delay is selected
        if (Tpath  = TIME'HIGH ) then
            -- Check if none of the condition for this output path are TRUE.
            -- Issue WARNING and schedule after 0 ns.
            assert FALSE report
             "No PathCondition TRUE for output path " & OutSignalName & ".  Using 0-delay"
             severity WARNING;
            Tpath := 0 ns;
        elsif (Tpath >= Tlast) then
            -- Use the delay from the most recently changed input signal
            Tpath := Tpath-Tlast;
        elsif (Tpath < Tlast) then
            -- Issue warning and schedule at the current time.
            assert FALSE report
             "Path length for " & OutSignalName & " is shorter than last input change. Using 0-delay"
             severity WARNING;
            Tpath := 0 ns;
        end if;
        -- Call VITAL glitch handler
        -- Assumption: It will be modified to take care of the Reject and XLimit
        --              cases. Else, we might have to write our own version of
        --              this procedure.

        if GlitchKind = OnEvent then
            VitalGlitchOnEvent (OutSignal, OutSignalName, GlitchData, OutTemp,
                               Tpath, GlitchMode, 0 ns);  -- GlitchDelay := 0 ns
        else
            VitalGlitchOnDetect (OutSignal, OutSignalName, GlitchData, OutTemp,
                               Tpath, GlitchMode, 0 ns); -- GlitchDelay := 0 ns
        end if;
END VitalPropagatePathDelay
```