VITAL

VHDL Initiative Toward ASIC Libraries

Model Development Specification

Version 2.0

Table of Contents

1.0 Purpose

The purpose of VITAL is to promote the rapid development and use of ASIC libraries for VHDL design. The purpose of this document is to provide the specification of VITAL compliant VHDL models and define the semantics of those models. This specification is an implementation of the requirements defined in the VITAL Technical Requirements Document.

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

The major elements of the specification are:

- VITAL timing routines and types
- A format for timing delay and constraint data
- A set of modeling conventions
- A (potentially acceleratable) primitive set
- An interface specification for models
- Example models to serve as a demonstration vehicle.

2.0 Introduction

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

“Accelerate the development of ASIC macrocell libraries by leveraging existing methodologies of model development”

From a simulation vendor’s perspective, this effort will also develop a de-facto standard for VHDL ASIC timing such that simulation vendors can optimize their tools around the standard in order to develop high-performance solutions.

2.1 Background

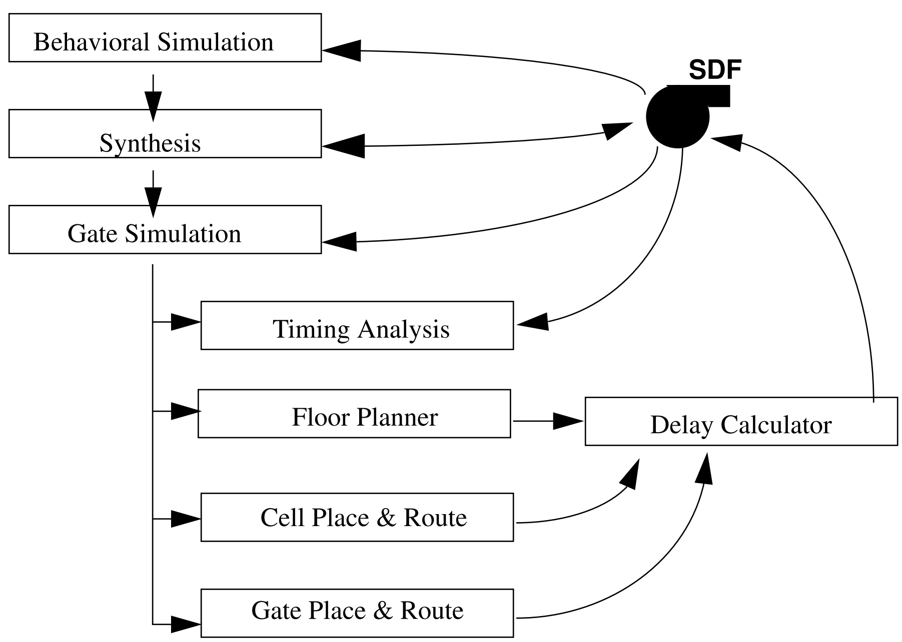
The VITAL effort germinated from ideas generated at the VHDL International Users’ Forum held in Scottsdale AZ in May 1992. Further discussions brought people to the conclusion that the biggest impediment to VHDL design was the lack of ASIC libraries; and that the biggest impediment to ASIC libraries was the lack of a uniform, efficient method for handling timing requirements 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 off of existing technology. VITAL has followed this approach using appropriate aspects of the Verilog timing methodology as its starting point including the SDF (Standard Delay File Format) as the starting point for a delay file format.This effort was aided by the fact that Verilog and SDF are public domain specifications.

2.2 VITAL Organization

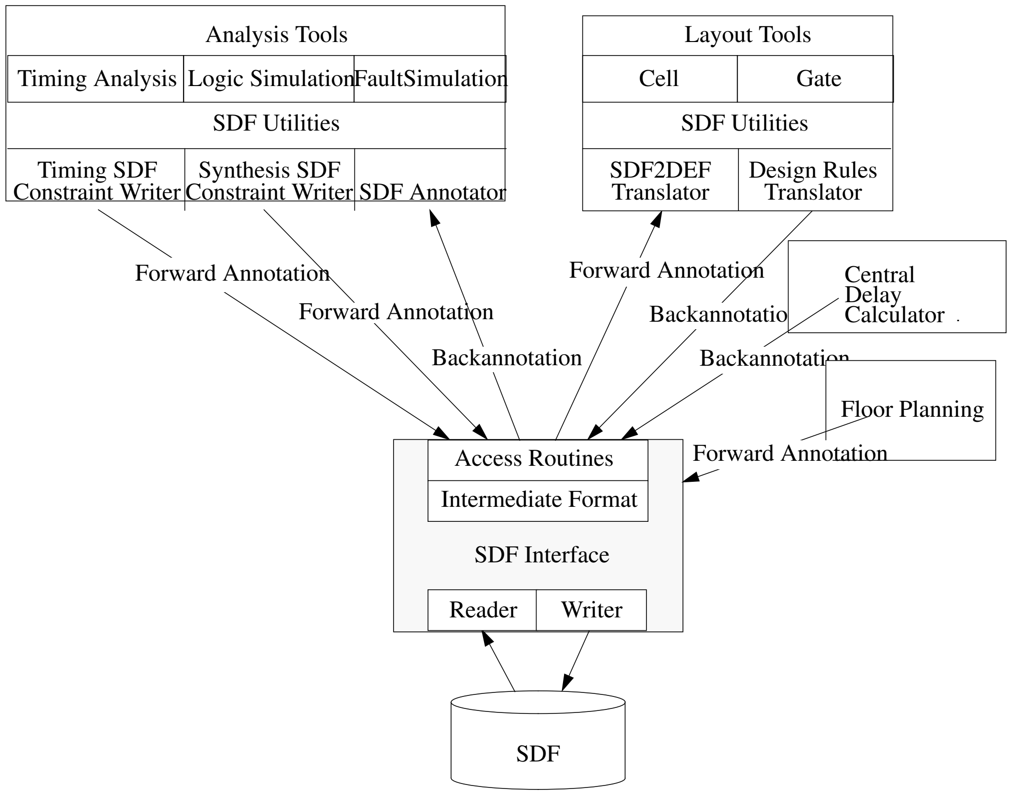
Over thirty companies have participated in the development of the VITAL technical plans. They have been led by Bill Billowitch of the VHDL Technology Group who serves as technical director; Steve Schulz of Texas Instruments Defense Systems and Electronics Group who serves as U.S. coordinator; and Eric Huyskens of Alcatel NV who serves as European coordinator. The member companies are: Alcatel NV, Bell Northern Research, CLSI/Solutions, Cadence Design Systems, DAZIX/Intergraph, Epson, Ericsson Telecom AB, Expertest, Fintronics, GenRad, Harris Semiconductor, Logic Modeling Corporation, Magneti Marelli Electronics, Matra MHS SA, Mentor Graphics, Motorola, National Semiconductor, NEC Electronics, Nextwave Design Automation, North Oaks EDA Consulting, OKI Semiconductor, Racal-Redac, S-MOS/Seiko, SGS Thomson, SimQuest, Sunrise Test Systems, Texas Instruments, Thomson CSF, Toshiba, Vantage Analysis Systems, VHDL Technology Group, Viewlogic and Xilinx.

2.3 Design Flow

While no specific design flow is implied, many of the discussions leading to the definition of this specification have centered around a style called Timing Driven Design. In this style information in an external timing file is used to communicate data and constraints between such tools as simulators, synthesizers, timing analyzers and place and route tools. Each tool either uses this data to perform its function or adds to or refines the data. This is illustrated in the following figure.



2.4 Design Environment



In this scenario, the delay calculator (and the SDF file) represents the interface between the designer, the tool suite and the ASIC vendor. Different calculations can be employed for different needs; e.g., pre-layout estimations and post-layout back annotation.

The separation of the timing characteristics of the device from the functional behavior allows a single library to be used for all processing tools in the design process. In addition, the same ASIC vendor library can be used for all the process technologies used by that vendor by supplying separate equations for the delay calculator. This dramatically simplifies the production and maintenance of these libraries for the ASIC vendor while providing more flexibility in choosing the process technology for the designer.

3.0 Strategy Summary

The goal established in the introduction above implicates a certain quality level regarding the accuracy of the library. This level is one of “Certified” libraries. For an ASIC foundry to “certify” a library means that the foundry will accept designs in the certified library’s format and that the timing information contained in the models is adequate for the purpose of a first cut timing analysis. Note however, that this level of accuracy may fall short of a “sign-off” grade simulation.

At the “sign-off” quality level, timing accuracy must be precise. Issues such as simultaneous events, glitch detection and propagation would be handled exactly as the foundry may require. Most foundries have invested considerable sums of money in developing an internal “golden” simulation environment, largely employing proprietary algorithms and techniques which have been developed internally.

The first phase of the VITAL initiative is geared at enabling the development of ASIC libraries at the Certified level. Subsequent phases may address the needs for sign-off grade simulation accuracy.Having said this, however, there are a number of language changes which would substantially enhance VHDL’s ability to become a sign-off grade simulator in the near future.

3.1 Scope

It is intended that this specification form the basis for defining the timing interfaces, timing routines and back-annotation mechanisms used in modeling ASIC libraries, ASIC’s and FPGA’s, as well as standard component models.

3.2 Dual Approach

ASIC libraries span the range of complexity from simple AND gates through to microprocessor core cells. This VITAL specification handles each region of complexity with a compatible, but slightly different modeling specification. At the macrocell level (i.e. AND gates, simple flops, etc.) a standard set of primitives is provided. Simulation vendors may optimize these primitives for performance knowing that a standard has been established. For complex models, a more generalized approach is specified.

Primitives: ASIC foundries and their customers expressed strong requirements for a solution which will yield high performance. Fast simulation and large capacity are critical issues. Under these conditions, tens and hundreds of thousands of elementary models (i.e. AND gates, flops) need to be simulated quickly and accurately. Rapid turn-around time for circuit changes need to be provided. The solution proposed for rapid simulation is to define a set of primitives which can be accelerated by the simulator vendor, but adhere to a standard semantic.

Megafunctions and complex cells: At the complex cell (i.e. 8051 core processor) level, the interaction between behavior and timing rarely lends itself to the techniques outlined for macrocells. Hence, the VITAL program developed an optimized model interface and guideline for the development of complex component models. The VITAL compliant component models have been developed to make use of SDF as do the Macrocell models.

3.3 General Philosophy

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

- Maximize efficiency and reduce the number of generic parameters to a minimum.
- Make the interface as intuitive as possible
- Allow modeling methodologies to improve in the future without significant changes in the interface.

A successful modeling methodology must first address the needs of its users and allow developers the flexibility to model in whatever fashion they require to achieve their objectives, yet be constraining only in the areas which require constraint for the common good. These goals bring us to the following requirements:

Optimized Primitives: A set of standard VITAL primitives are defined which can be accelerated by the simulator vendor to achieve high performance simulation. These models will use the same interface as provided with more complex cells modeled according to the VITAL guidelines.

Observed Controllability: The control of a model, both in the functional and timing sense must be observable by constructs used within the model. In other words, the semantics of how a model operates must be defined by the model's interface, and how those interface objects are used within the model written in VHDL. If there are any "back-door" techniques employed with the objectives of efficiency in mind, then those same techniques shall be capable of being expressed in VHDL regardless of whether they are actually expressed as such.

Timing Calculations Done Off Chip: All timing calculations, with the exception of delay selection, shall be conducted "off-chip". This relieves the model from having to perform its own delay calculations based upon environmental conditions and loading. By simply loading the pre-calculated timing parameters into the model, significant savings are realized in model complexity and efficiency.

Simplicity and Extensibility: The required model interface shall be as simple as possible and allow the modeler to utilize the full richness of the language if so desired.

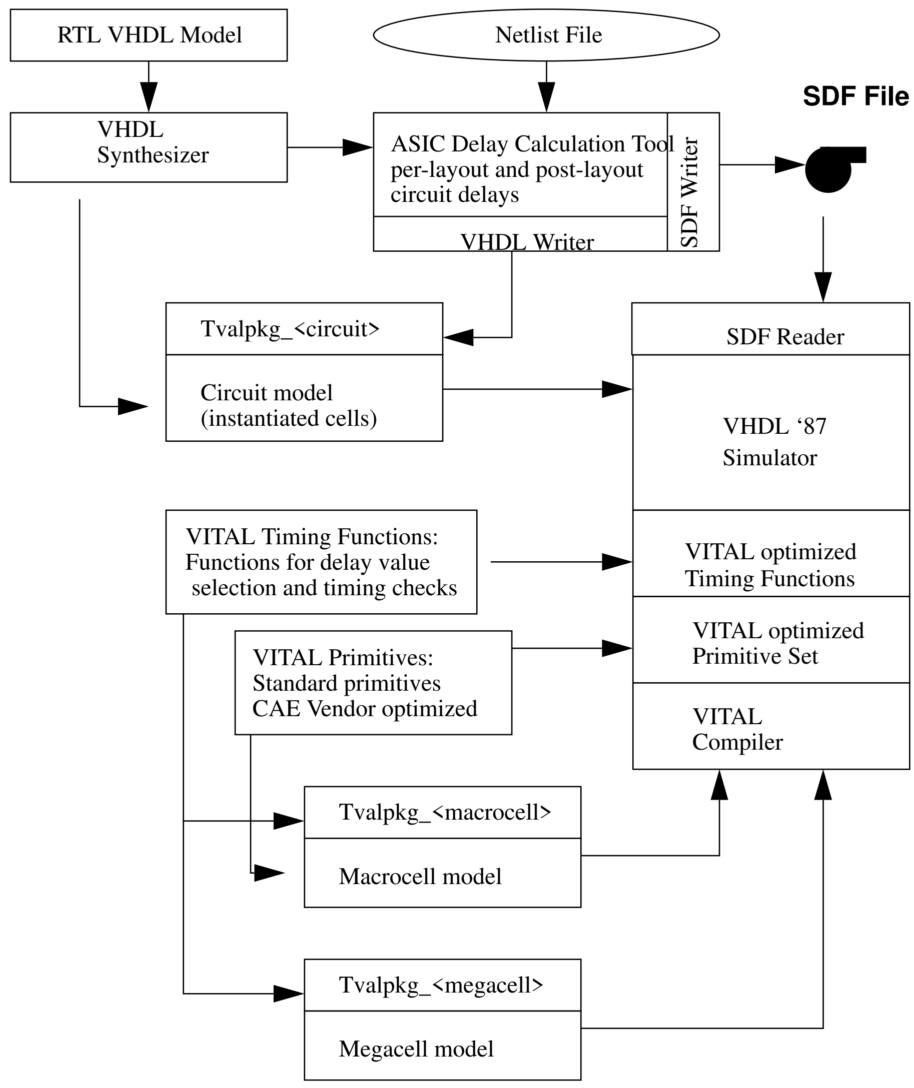
4.0 VITAL Basics

The illustration below demonstrates how timing information flows in a typical VITAL compliant environment.

The VITAL specification defines:

- (a) the timing interface of a VHDL model or VHDL circuit (connected list of instantiated models),
- (b) the mapping of SDF information into an equivalent VHDL timing data package,
- (c) timing functions which can be accelerated by a simulator vendor
- (d) a set of acceleratable primitives.

An ASIC Delay Calculation Tool, normally provided by the ASIC foundry or simulation vendor, will generate an SDF data file which represents the derated timing of a circuit. This delay calculator can also produce a VHDL timing data package which can contain essentially the same information as the SDF file.



Simulator vendors may certainly compile the standard VHDL which is generated by the delay calculator. However, vendors will also have the prorogative to implement an SDF import capability to allow SDF data insertion in the simulation runtime kernel as if it were compiled through standard VHDL compilation mechanisms.

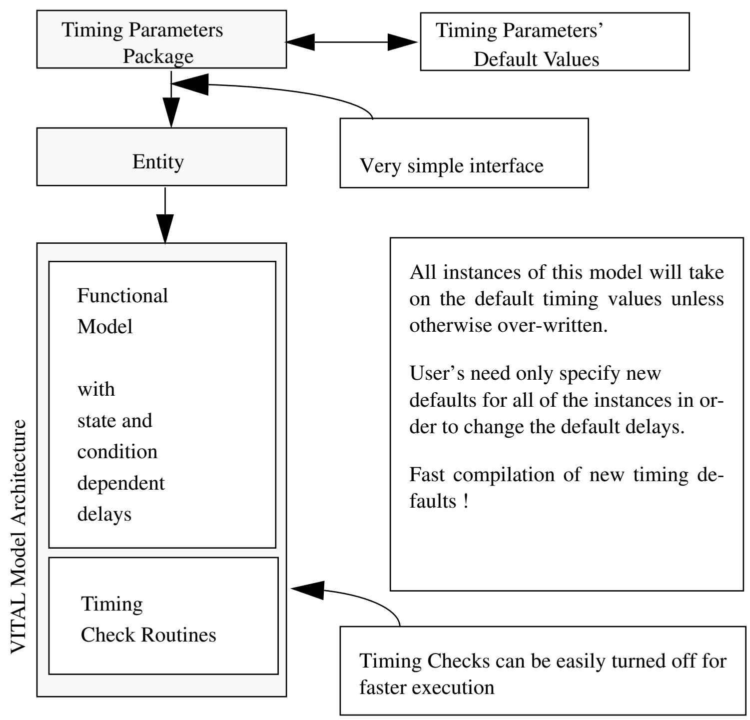
Classical simulation environments employ built-in primitives which are optimized for runtime speed. VITAL defines a set of standard primitives for this purpose.

Since most models call routines to handle timing checks and process timing data, it would be beneficial to standardize on a set of acceleratable routines which could form the basis for developing timing accurate models.

5.0 VITAL VHDL Timing Interface

In developing a model interface, there were a few major objectives considered:

- Maximize efficiency and reduce the number of generic parameters to a minimum.
- Make the interface as intuitive as possible



Under the VITAL recommendation, the model maintains a simple interface while allowing back-annotation tools to provide updated timing information without requiring any modifications to the model. In addition, the methodology has the following benefits:

Timing Calculations Done Off Chip: All timing calculations, with the exception of delay selection, shall be conducted "off-chip". This relieves the model from having to perform its own delay calculations based upon environmental conditions and loading. By simply loading the pre-calculated timing parameters into the model, significant savings are realized in model complexity and efficiency.

Maintainability: The timing model and functional models may be separately maintained and developed.

5.1 Generic Parameters

Generic parameters define the interface to the model for non-port related information. The generic parameter list has been designed to be simple, yet flexible and powerful. Included in the generic parameter list are values pertaining to:

- Model Control: Such as turning off timing checks and selecting min-typ-max timing values.
- Timing Values: All timing values are passed into the model through an array data structure. The values of the array are maintained in a separate timing package

```
GENERIC (

-------------------------------------------------------------
-- System wide defaults are established. To override the
-- default, simply associate your own values into the
-- generic map of the instance or configuration of this
-- component where you wish a change to take place.
-------------------------------------------------------------
-- The default timing values are contained in the Tvalpkg_<modelname> package.
-------------------------------------------------------------
    Tval            : TvalType      := Tdefault_<modelname>;
    TimeMode        : TimeModeType  := t_typical;
    TimingChecksOn  : Boolean       := TRUE

);
```

- **Tval**: Tval is a data structure which contains all of the timing parameters used by the model. Tval's value is originally set by its default value (Tdefault_<modelname>). The data structure designed to support Min-Typ-Max timing is a simply array of three TIME values. A larger, integer index array of this Min-Typ-Max array is created to accommodate as many timing parameters as needed by the model. The result is an integer index array of arrays of time. The benefit of this data structure are many: (a) a common type can be created to accommodate an array of any length (i.e. an arbitrarily large number of timing parameters can be accommodated, (b) the model interface is a simple single parameter (TVAL) and (c) all of the actual data is stored in package bodies which can be accessed using "back-door" generic loading techniques common among commercial VHDL simulators [Vantage, Cadence].
- **TimeMode**: Allows the model to select between t_minimum, t_typical, and t_maximum timing values using first generation modeling techniques.
- **TimingChecksOn**: A flag which instruct the model to enable timing checking.

With this interface, all instances of a given model will be provided a set of default timing values which can be over-written on an instance by instance basis if desired.

`NOTE: regardless of the number of timing parameters needed, the generic interface never becomes more complex than shown.`

5.2 Types used in the Generic Parameter Interface

Keeping with the philosophy of utilizing existing practices to accelerate the availability of ASIC macrocell libraries, the following types have been contributed, with permission, from the Std_Timing package offered by the VHDL Technology Group.

MinTypMax: Array of time containing minimum, typical and maximum values.

```
TYPE TimeModeType IS ( t_minimum, -- choose minimum time spec
                       t_typical, -- choose typical time spec
                       t_maximum, -- choose maximum time spec
                       t_special  -- choose user's time spec
                     );

  TYPE MTM is array (TimeModeType range <>) of TIME;
  Subtype MinTypMax is MTM (t_minimum to t_maximum);
   -----------------------------------------------------------------------
   -- Example ( generic declarations ) :
   --  tplh_a2_y2 : MinTypMax := (12.0 ns, 17.0 ns, 32.0 ns);
   --                           ^^^     ^^^     ^^^
   --                           min     typ     max
   -----------------------------------------------------------------------
```

TvalType: Data structure used to hold all timing values

```
  TYPE TvalType is array (Natural range <>) of MinTypMax;
  -----------------------------------------------------------------------
  -- Example
  -- CONSTANT Tval_myinstance: TvalType(1 to 20) := (others => (0 ns, 0 ns, 0 ns));
  -----------------------------------------------------------------------
```

5.3 VHDL Timing Package

VITAL models must follow the following guidelines:

- VITAL models must have a timing data package associated with the model. The name of this package must be Tvalpkg_<modelname> where <modelname> is the name of the model's entity.
- Within the package_declarative_part of the package named Tvalpkg_<modelname>, a set of integer constants must be declared. These constants define indices into an array and at the same time, the names of the constants should refer to the timing specification names and be consecutively numbered (i.e. CONSTANT tplh_clk_q : integer := 1;).

```
Example:

PACKAGE Tvalpkg_scanff IS
  CONSTANT tplh_CLK_Q     : INTEGER := 1;
  CONSTANT tphl_CLK_Q     : INTEGER := 2;
  CONSTANT tphl_CLR_Q     : INTEGER := 3;

  -- ...additional list of constants

  CONSTANT trecoveryl_SET_CLK   : INTEGER := 28;
  CONSTANT t29                  : INTEGER := 29;
  CONSTANT t30                  : INTEGER := 30;
  -- ---------------------------------------------------------------------------------
  -- Deferred Constant Array
  -- ---------------------------------------------------------------------------------
  CONSTANT Tdefault_scanff : TvalType (1 to 30);
END Tvalpkg_scanff;
```

- Within the package_declarative_part of the package named Tvalpkg_<modelname>, an array named Tdefault_<modelname> must be declared. The dimensionality of this array should start with the index 1 and span the full list of integer constant which was just previously declared. In other words, if 30 integer constants were declared, corresponding to 30 timing parameters, the array would be declared as: CONSTANT Tdefault_<modelname> : TvalType(1 to 30); The array should defer its values to the package body.
- VITAL models must declare a package body corresponding to the name Tvalpkg_<modelname> as described above. Within this package body, the deferred constant values of the Tdefault_<modelname> array must be defined. An example follows:

```
Example:

PACKAGE BODY Tvalpkg_scanff IS

  CONSTANT Tdefault_scanff : TvalType(1 to 30) := (
--                        min       typ     max
         tplh_CLK_Q  => (  16.5 ns,  16.5 ns,  16.5 ns ),
         tphl_CLK_Q => (     5 ns,     5 ns,     5 ns ) ,
         tphl_CLR_Q => (     5 ns,     5 ns,     5 ns ),
...
         t29        => (    30 ns,    30 ns,    30 ns ),
         t30        => (    30 ns,    30 ns,    30 ns )
  );
END Tvalpkg_scanff;
```

6.0 VITAL VHDL Common Specifications

6.1 VITAL Restrictions

A VITAL model shall not contain any reference to a global signal.

6.2 VITAL Port Interface DataType

All ports of the model shall utilize the Std_Logic_1164 datatype. The values of the datatype are:

‘U’ . . . . .Uninitialized
‘X’ . . . . .Forcing Unknown
‘0’. . . . . .Forcing Low
‘1’. . . . . .Forcing High
‘Z’ . . . . .High Impedance
‘W’ . . . . .Weak Unknown
‘L’. . . . . .Weak Low
‘H’ . . . . .Weak High
‘-’ . . . . .don’t care

VITAL model ports shall use the Std_logic_1164 data type or subtypes. [IEEE Std 1164].

Scalar Ports: Models shall use the std_logic datatype or a subtype thereof.
Array Ports: Models shall use the std_logic_vector datatype. No subtypes are allowed.

Optimization: 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.

The following rationale supports this specification:

In VHDL, we are forced to consider signal value resolution. Yet the language treats an array of a resolved type as unique from an array of the same unresolved type. Consequently, the designer must be tediously aware of whether to model a connection as a resolved type or unresolved type particularly in the case of arrays (i.e. data buses). The worst case, from an efficiency perspective is to use resolved signals. However, from a modeling perspective, there are a number of advantages. Fortunately, there are techniques which can be used to optimize simulation tools in their handling of resolved types.:

TABLE 1. Pros and Cons of using only resolved data types

	PROS	CONS
Interconnections of Components	Significantly eases the connectability of components. Alternative would require the designer to tag every net as being resolved or unresolved	
Multiple Driver Detection		Accidentally connected multiple drivers not found by the compiler
Efficiency	Can be optimized.	Forces the simulator to optimize the network to recognize single driver resolved nets
Type Conversions	This technique eliminates the need for type conversions on array resolved ports to connect to unresolved array ports	

The issue of detecting accidentally multiply driven nets is the only issue which is of real concern to a digital designer using the models, with respect to signal resolution. As a compromise, a suggestion could be made for a simulator vendor to run a simple design rule check on the netlist and indicate each of the nets which are multiply driven so that the designer is made aware of the potential conflict areas.

6.3 VITAL Attribute

VITAL architectures will be marked with the attribute ‘VITAL so that tools can easily recognize them and check for compliance to these restrictions. A VHDL simulator which does not support VITAL may process a VITAL model the same way a non-VITAL model is processed.

VITAL compliant simulators have the attribute ‘VITAL predefined. When recognizing the attribute the simulator will:

- check the model for compliance
- check for the existence of a timing file

The timing file may be designated as an implementation dependent string.

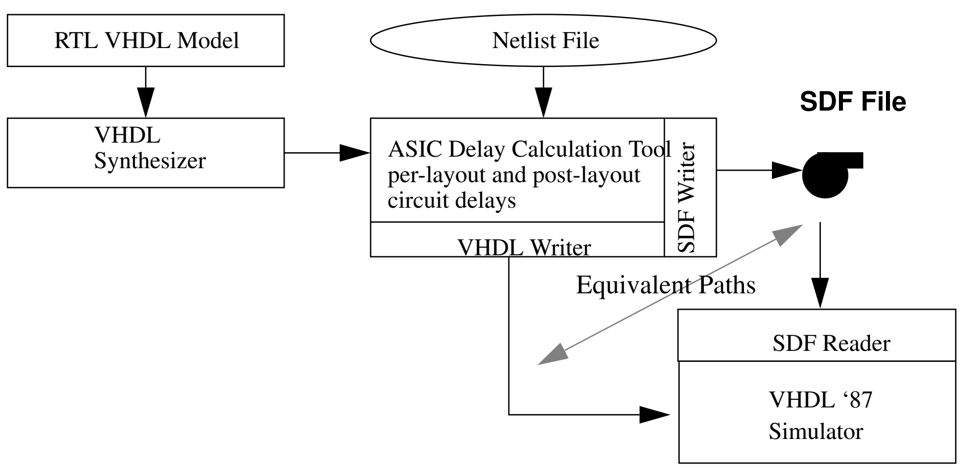
7.0 VITAL SDF Timing Interface

VHDL has weaknesses in providing topological information to the design hierarchy. Therefore, delay calculations based upon circuit loading and other factors cannot be readily handled. As a result, most vendors and users provide alternative mechanisms for back-annotating delay information when VHDL models are concerned. While these varied techniques may work, they all represent different ways of approaching the problem. A standard delay representation would provide a basis for interoperable back-annotation tool sets. For this, the VITAL initiative proposes the SDF standard.

The OVI (Open Verilog International) Standard Delay File (SDF) will form the basis for the implementation of Back-Annotation in VHDL. There are a number of ways in which timing data can be read by the simulator and associated with a given model instance:

- A delay calculator may directly generate a VITAL compliant timing data package which contains the timing data values required by the model. Instance specific timing information can then be associated with the model via Configuration Declarations.
- A delay calculator may directly generate an SDF file. A VITAL compliant simulator, having implemented an SDF reader, could then directly import timing data into the placeholders established by the prior compilation of the VHDL timing data package.

Note, however, that other methods, such as reading VHDL source code via TEXTIO were considered, but rejected because of efficiency considerations. In addition, VHDL source code procedural interfaces to SDF were discounted as well because of their relative poor performance.



7.1 Back-Annotation via VITAL compliant timing package

VITAL models must declare a corresponding VHDL timing package called Tvalpkg_<modelname>. This package will contain the declarations of the timing parameters used by the model.

A delay calculator may update the values of the VHDL Tvalpkg. Then by recompiling the VHDL timing data package, new values will be provided for the model. While this technique may work adequately well for single models, it fails to provide an efficient solution for a typical ASIC design which may contain

25000 components, each with its own timing data values. In such cases, a more efficient approach is called upon.

7.2 Back-Annotation via SDF Direct Import

For high gate count designs, the compilation of a VITAL timing data package containing possibly hundreds of thousands of lines of VHDL code would be a daunting endeavor for most tool providers not to mention the compilation time required. Clearly an alternative approach is preferred which could accelerate the loading of back-annotation data.

As shown in the illustration, once a VHDL timing package has been compiled and a path and placeholder for timing data has been established, subsequent modifications of the timing data can be accomplished in either of two ways:

- Modification of the VHDL timing data package body and recompilation of the package body, or...
- Direct import of the equivalently represented SDF data values.

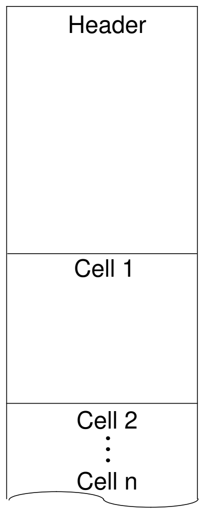
This hybrid approach allows the developer maximum flexibility in representing and importing back-annotation data into a design or a model. The rules of compatibility follow:

- If the SDF contains information not specified in the Tvalpkg, the SDF information item will be ignored.
- If the SDF contains less information than provided by the Tvalpkg, then for those data values not provided by the SDF, the default values provided by the VHDL Tvalpkg will prevail.

7.3 Correspondence of SDF Information to VHDL Design

The SDF information, containing the timing and constraint data for a design, is tied to the design through cell and path names. Cell names correspond to component instance labels and/or pathnames. An instance name of “*” indicates that all instances take on the same timing values.

.




```
(DELAYFILE

        (DESIGN      "my_design")
        (DATE        "Jun 12, 1992 09:46")
        (VENDOR      "Southwestern ASIC")
        (PROGRAM     "Fast program")
        (VERSION     "1.2a")
        (DIVIDER     /)
        (VOLTAGE     4.5:5.0:5.5)
        (PROCESS     "best")
        (TEMPERATURE 27:55:100)
        (TIMESCALE   100 ps)

(CELL (CELLTYPE "DFF")
        (INSTANCE a.b.c)
        (DELAY
                (IOPATH (posedge clk) q (2:3:4) (5:6:7))
                (PORT clr (2:3:4) (5:6:7))
        )
)
```

SDF to VHDL Mapping : Details of the SDF to VHDL mapping can be found in the Appendix.

8.0 VITAL VHDL Timing Functions

A core set of timing functions are provided as part of the VITAL standard. In keeping with the philosophy of utilizing existing practices to accelerate the availability of ASIC macrocell libraries, the following subprogram declarations have been contributed, with permission, from the Std_Timing package offered by the VHDL Technology Group.

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 to 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 setup 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.

SetupViolation . . . . Checks for Setup Time Violations

PURPOSE:

To check for a setup violation.

DECLARATION:

```
Function setupviolation (
    Signal      TestPort      : IN    std_ulogic;
    Constant    TestPortName  : IN    STRING := “”;
    Signal      RefPort       : IN    std_ulogic;
    Constant    RefPortName   : IN    STRING := “”;
    Constant    t_setup_hi    : IN    TIME := 0 ns;
    Constant    t_setup_lo    : IN    TIME := 0 ns;
    Constant    condition     : IN    BOOLEAN;
    Constant    HeaderMsg     : IN    STRING := “”
) return BOOLEAN;
```

DESCRIPTION:

t_setup_hi ::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘1’.

t_setup_lo ::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘0’.

ASSUMPTIONS:

t_setup_hi and t_setup_lo must both be non-negative numbers. If negative setup time specifications are expected, then the combined “TimingViolation” function should be used which has been designed to accommodate negative setup and hold times.

BUILT IN ERROR TRAPS:

If the observed setup time is less than the specified values, an assertion message will be issued and the function will return a TRUE value. Otherwise, no message will be asserted and the function will return a FALSE value.

EXAMPLES:

```
If setupviolation (     testport        => D,
                        testportname    => “D”,
                        refport         => CLK,
                        refportname     => “CLK”,
                        t_setup_hi      => 22.5 ns,
                        t_setup_lo      => 21.0 ns,
                        condition       => rising_edge(clk),
                        headerMsg       => “/u1/u23/u224”
                    ) then
        Q <= ‘X’;
else
        Q <= ‘Z’;
end if;
```

SetupCheck . . . . Checks for Setup Time Violations

PURPOSE:

To check for a setup violation.

DECLARATION:

```
Procedure SetupCheck (
      Signal      TestPort       :  IN     std_ulogic;
      Constant    TestPortName   :  IN     STRING := “”;
      Signal      RefPort        :  IN     std_ulogic;
      Constant    RefPortName    :  IN     STRING := “”;
      Constant    t_setup_hi     :  IN     TIME := 0 ns;
      Constant    t_setup_lo     :  IN     TIME := 0 ns;
      Constant    condition      :  IN     BOOLEAN;
      Constant    HeaderMsg      :  IN     STRING := “”
);
```

DESCRIPTION:

t_setup_hi	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘1’.
t_setup_lo	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘0’.

ASSUMPTIONS:

t_setup_hi and t_setup_lo must both be non-negative numbers. If negative setup time specifications are expected, then the user should use the combined “TimingViolation” function which has been designed to accommodate negative setup and hold times.

BUILT IN ERROR TRAPS:

If the observed setup time is less than the specified values, an assertion message will be issued. Otherwise, no message will be asserted.

EXAMPLES:

```
 setupcheck (          testport         => D,
                       testportname     => “D”,
                       refport          => CLK,
                       refportname      => “CLK”,
                       t_setup_hi       => 22.5 ns,
                       t_setup_lo       => 21.0 ns,
                       condition        => rising_edge(clk),
                       headerMsg        => “/u1/u23/u224”
);
```

HoldViolation . . . . Checks for Hold Time Violations

PURPOSE:

To check for a hold time violation.

DECLARATION:

```
Function HoldViolation (
    Signal      TestPort        :  IN std_ulogic;
    Constant    TestPortName    :  IN STRING := “”;
    Signal      RefPort         :  IN std_ulogic;
    Constant    RefPortName     :  IN STRING := “”;
    Constant    t_hold_hi       :  IN TIME := 0 ns;
    Constant    t_hold_lo       :  IN TIME := 0 ns;
    Constant    condition       :  IN BOOLEAN;
    Constant    HeaderMsg       :  IN STRING := “”
) return BOOLEAN;
```

DESCRIPTION:

t_hold_hi	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘1’.
t_hold_lo	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘0’.

ASSUMPTIONS:

t_hold_hi and t_hold_lo must both be non-negative numbers. If negative hold time specifications are expected, then the user should use the combined “TimingViolation” function which has been designed to accommodate negative setup and hold times.

BUILT IN ERROR TRAPS:

If the observed hold time is less than the specified values, an assertion message will be issued and the function will return a TRUE value. Otherwise, no message will be asserted and the function will return a FALSE value.

EXAMPLES:

```
If holdviolation (      testport        => D,
                        testportname    => “D”,
                        refport         => CLK,
                        refportname     => “CLK”,
                        t_hold_hi       => 22.5 ns,
                        t_hold_lo       => 21.0 ns,
                        condition       => (clk = ‘1’),
                        headerMsg       => “/u1/u23/u224”
                    ) then
        Q <= ‘X’;
else
        Q <= ‘Z’;
end if;
```

HoldCheck .... Checks for Hold Time Violations

PURPOSE:

To check for a hold time violation.

DECLARATION:

```
Procedure HoldCheck (
    Signal      TestPort        : IN std_ulogic;
    Constant    TestPortName    : IN STRING := “”;
    Signal      RefPort         : IN std_ulogic;
    Constant    RefPortName     : IN STRING := “”;
    Constant    t_hold_hi       : IN TIME := 0 ns;
    Constant    t_hold_lo       : IN TIME := 0 ns;
    Constant    condition       : IN BOOLEAN;
    Constant    HeaderMsg       : IN STRING := “”
);
```

DESCRIPTION:

t_hold_hi ::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘1’.

t_hold_lo ::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘0’.

ASSUMPTIONS:

t_hold_hi and t_hold_lo must both be non-negative numbers. If negative hold time specifications are expected, then the user should use the combined “TimingViolation” function which has been designed to accommodate negative setup and hold times.

BUILT IN ERROR TRAPS:

If the observed hold time is less than the specified values, an assertion message will be issued. Otherwise, no message will be asserted.

EXAMPLES:

```
holdcheck (         testport        => D,
                    testportname    => “D”,
                    refport         => CLK,
                    refportname     => “CLK”,
                    t_hold_hi       => 22.5 ns,
                    t_hold_lo       => 21.0 ns,
                    condition       => (clk = ‘1’),
                    headerMsg       => “/u1/u23/u224”
);
```

TimingViolation .... Checks for Setup and Hold Violations

PURPOSE:

To check for a setup time or a hold time violation.

DECLARATION:

```
Function TimingViolation (
    Signal      TestPort        : IN std_ulogic;
    Constant    TestPortName    : IN STRING := "";
    Signal      RefPort         : IN std_ulogic;
    Constant    RefPortName     : IN STRING := "";
    Constant    t_setup_hi      : IN TIME := 0 ns;
    Constant    t_setup_lo      : IN TIME := 0 ns;
    Constant    t_hold_hi       : IN TIME := 0 ns;
    Constant    t_hold_lo       : IN TIME := 0 ns;
    Constant    condition       : IN BOOLEAN;
    Constant    HeaderMsg       : IN STRING := ""
) return BOOLEAN;
```

DESCRIPTION:

t_hold_hi	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is '1'.
t_hold_lo	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is '0'.
t_setup_hi	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is '1'.
t_setup_lo	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is '0'.

ASSUMPTIONS:

t_setup_hi, t_setup_lo, t_hold_hi and t_hold_lo may be either positive or negative. Negative setup times are treated as positive hold times and negative hold times are treated as positive setup times.

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 function will return a TRUE value. Otherwise, no message will be asserted and the function will return a FALSE value.

EXAMPLES:

```
If timingviolation (    testport     => D,            testportname   => "D",
                        refport      => CLK,          refportname    => "CLK",
                        t_setup_hi   => 32.0 ns,      t_setup_lo     => 22.1 ns,
                        t_hold_hi    => 22.5 ns,      t_hold_lo      => 21.0 ns,
                        condition    => (clk = '1'),  headerMsg      => "/u1/u23/u224"
                    ) then
        Q <= 'X';
else
        Q <= 'Z';
end if;
```

TimingCheck . . . . Checks for Setup and Hold Violations

PURPOSE:

To check for a setup time or a hold time violation.

OVERLOADED DECLARATIONS:

```
Procedure TimingCheck (
      Signal      TestPort        : IN std_ulogic;
      Constant    TestPortName    : IN STRING := “”;
      Signal      RefPort         : IN std_ulogic;
      Constant    RefPortName     : IN STRING := “”;
      Constant    t_setup_hi      : IN TIME := 0 ns;
      Constant    t_setup_lo      : IN TIME := 0 ns;
      Constant    t_hold_hi       : IN TIME := 0 ns;
      Constant    t_hold_lo       : IN TIME := 0 ns;
      Constant    condition       : IN BOOLEAN;
      Constant    HeaderMsg       : IN STRING := “”;
      Variable    Violation       : INOUT Boolean
) ;
```

DESCRIPTION:

t_hold_hi	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘1’.
t_hold_lo	::= specification of the minimum interval after the triggering edge of the referenced signal when the testport value is ‘0’.
t_setup_hi	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘1’.
t_setup_lo	::= specification of the minimum interval preceding the triggering edge of the referenced signal when the testport value is ‘0’.

ASSUMPTIONS:

t_setup_hi, t_setup_lo, t_hold_hi and t_hold_lo may be either positive or negative. Negative setup times are treated as positive hold times and negative hold times are treated as positive setup times.

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..

EXAMPLES:

```
PROCESS (clk, D)
        VARIABLE Violation : BOOLEAN := False;
BEGIN
    TimingCheck (
                testport        => D,           testportname    => "D",
                refport         => CLK,         refport_name    => "CLK",
                t_setup_hi      => 23 ns,       t_setup_lo      => 23 ns,
                t_hold_hi       => 5 ns,        t_hold_lo       => 6 ns,
                condition       => (CLK = '1'), header_msg      => InstanceHeader,
                violation       => Violation    );
        If Violation THEN       Q <= 'X';  ELSE                 Q <= 'Z';  END IF;
 END PROCESS;
```

PeriodCheck . . . . Checks for Periodicity Violations

PURPOSE:

To check for periodicity violations

DECLARATION:

```
Procedure PeriodCheck (
    Signal      TestPort        : IN std_ulogic;
    Constant    TestPortName    : IN STRING := “”;
    Constant    RefPort         : IN std_ulogic := ‘-’;
    Constant    RefPortName     : IN STRING := “”;
    Constant    PeriodMin       : IN TIME := 0 ns;
    Constant    PeriodMax       : IN TIME := TIME’HIGH;
    Constant    pw_hi_min_hi    : IN TIME := 0 ns; -- hi PW when ref = hi
    Constant    pw_hi_min_lo    : IN TIME := 0 ns; -- hi PW when ref = lo
    Constant    pw_hi_max       : IN TIME := TIME’HIGH;
    Constant    pw_lo_min_hi    : IN TIME := 0 ns; -- lo PW when ref = hi
    Constant    pw_lo_min_lo    : IN TIME := 0 ns; -- lo PW when ref = lo
    Constant    pw_lo_max       : IN TIME := TIME’HIGH;
    Variable    info            : INOUT PeriodCheckInfoType := PeriodCheckInfo_Init;
    Variable    Violation       : OUT BOOLEAN;
    Constant    HeaderMsg       : IN STRING := “”
);
```

DESCRIPTION:

This procedure monitors the test port and determines if the test port violates any of the timing specifications. Info is used to hold timing information and should be used as shown in the example.

BUILT IN ERROR TRAPS:

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

EXAMPLES:

```
check_clk : PROCESS (clk)
        variable timekeeper : PeriodCheckInfoType := PeriodCheckInfo_Init;
        variable violation : boolean := false;
BEGIN
        periodcheck (
                testport        =>      CLK,                    testportname    => “CLK”,
                refport         =>      OPEN,                   refportname     => “”,
                periodmin       =>      50 ns,                  periodmax       => 100 ns,
                pw_hi_min_hi    =>      10 ns,                  pw_hi_min_lo    => 10 ns,
                pw_hi_max       =>      50 ns,                  pw_lo_min_hi    => 10 ns,
                pw_lo_min_lo    =>      10 ns,                  pw_lo_max       => 50 ns,
                info            =>      timekeeper,             violation       => violation,
                HeaderMsg       =>      InstancePathName        );
        if violation then
                Q <= ‘X’;
        else
                Q <= ‘Z’;
        end if;
END PROCESS;
```

PulseCheck .... Checks for Minimum Pulse Width

PURPOSE:

To check for the minimum pulse width.

DECLARATION:

```
Procedure PulseCheck (
    Signal      TestPort        : IN std_ulogic;
    Constant    TestPortName    : IN STRING := “”;
    Constant    t_pulse_hi      : IN TIME := TIME’HIGH;
    Constant    t_pulse_lo      : IN TIME := TIME’HIGH;
    Variable    timemark        : INOUT TIME;
    Constant    HeaderMsg       : IN STRING := “”
);
```

DESCRIPTION:

This procedure tests for pulses of less than the specified width which occur on the test signal.

A pulse is defined as two EVENTS which occur in close time proximity to one another such that the time interval between events is less than some specified time.

t_pulse_hi ::= minimum allowable time between the last time the test port changed and the current time when the new test port value is ‘1’.

t_pulse_lo ::= minimum allowable time between the last time the test port changed and the current time when the new test port value is ‘0’.

ASSUMPTIONS:

t_pulse_hi and t_pulse_lo must both be non-negative numbers.

BUILT IN ERROR TRAPS:

This procedure will issue an assertion message if the test port violates the pulse width specifications.

EXAMPLES:

```
check_clk : PROCESS (clk)

    variable timemark : time;

BEGIN
    pulsecheck (
        TestPort        =>  CLK,
        TestPortName    =>  “CLK”,
        t_pulse_hi      =>  10 ns,
        t_pulse_lo      =>  10 ns,
        timemark        =>  timemark,
        HeaderMsg       =>  InstancePathName
        );
END PROCESS;
```

CalcDelay .... Calculate Output Delays

PURPOSE:

To calculate the output delays based upon the signal values.

OVERLOADED DECLARATIONS:

```
Function CalcDelay (
    Constant    newval      :  IN std_ulogic;   -- new value of signal
    Constant    oldval      :  IN std_ulogic;   -- old value of signal
    Constant    tp01        :  IN TIME := 1 ns;    -- 0->1 delay value
    Constant    tp10        :  IN TIME := 1 ns;    -- 1->0 delay value
    Constant    tp0z        :  IN TIME := 1 ns;    -- 0->z delay value
    Constant    tp1z        :  IN TIME := 1 ns;    -- 1->z delay value
    Constant    tpz0        :  IN TIME := 1 ns;    -- z->0 delay value
    Constant    tpz1        :  IN TIME := 1 ns    -- z->1 delay value
) return TIME;
```

```
Function CalcDelay (
    Constant    value       :  IN std_ulogic;   -- new value of signal
    Constant    Tp1         :  IN TIME := 1 ns;    -- 0->1 delay value
    Constant    Tp0         :  IN TIME := 1 ns    -- 1-> 0 delay value
) return TIME;
```

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 three parameter versions, only the newly assigned value is used in determining the delay.

DEFAULTS:

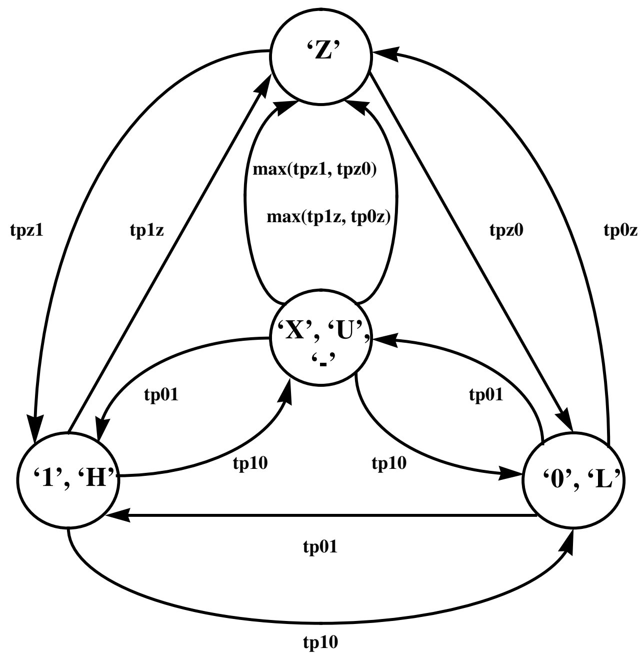
For the verbose form, not all of the parameters need to be passed since defaults are provided for those not passed.

BUILT IN ERROR TRAPS:

None.

RESULT:

The delay that is returned by this function is described by the state diagram to follow: In the diagram each node represents a logic value. Each arc represents a transistion from one logic value to another. The delay associated with each arc is the value that is returned by the function for that transition. Note that for the three parameter versions of this function the delay is a function of only the newly assigned value of the signal.



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
                CalcDelay    (  newval      => the_new_value,
                                oldval      => '-',
                                tp01        => tplh_a_y  ,
                                tp10        => tphl_a_y
                                -- in this example the remaining
                                -- parameters are not specified
                             );
```

9.0 VITAL VHDL Primitives

There are generally two styles of modeling ASIC libraries. The first is to model the cell from a behavioral and timing perspective. The second is to construct the model by interconnecting predefined primitives. With properly designed primitives, acceleration techniques could be employed to provide high performance simulation for modelers who prefer to use “built-in” primitives.

The development of a full set of primitives is well beyond the scope of this effort. However, a small number of primitives will be defined and developed to illustrate the interfaces required for building additional primitives according to the guidelines described by VITAL.

9.1 N-input Logical Gates

Any # of input ports, one output port

<GateName> ::= AND, NAND, OR, NOR, XOR, XNOR

```
------------------------------------------------------------------------------------------------
ENTITY VITAL_<GateName> IS
  GENERIC (
      ---------------------------------------------------------------
      -- Timing Data
      ---------------------------------------------------------------
      Tval             : TvalType       := VitalGateDelay2;  -- default is set for a 2-input gate
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      Size             : Natural        := VitalGateSize2;
      InstancePath     : string         := ""                -- instance pathname
  );
  PORT (
      Data     : IN  std_logic_vector(0 to (Size-1));     -- Input data to the gate
      Result   : OUT std_logic                            -- AND of all "Data" subelements
  );
END VITAL_<GateName>;
```

9.2 Buffer and Inverter

Non-inverting and Inverting, single input, single output

Buffer

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_INV IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalGateDelay1;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Result   : OUT std_logic  -- Inversion of the Data port
  );
END VITAL_INV;
```

Inverter

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_BUF IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalGateDelay1;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Result   : OUT std_logic  -- Buffered copy of the Data port
  );
END VITAL_BUF;
```

9.3 Three-State Drivers

Non-inverting and Inverting three-state drivers with active high or active low enables

Buffer with active low Enable

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_BUFIF0 IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalBUFFERdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Enable   : IN  std_logic; -- Data -> Result if Enable = '0'
      Result   : OUT std_logic  -- Buffered copy of the Data port
  );
END VITAL_BUFIF0;
```

Buffer with active high Enable

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_BUFIF1 IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalBUFFERdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Enable   : IN  std_logic; -- Data -> Result if Enable = '1'
      Result   : OUT std_logic  -- Buffered copy of the Data port
  );
END VITAL_BUFIF1;
```

Inverter with active low Enable

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_INVIF0 IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalBUFFERdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Enable   : IN  std_logic; -- ~Data -> Result if Enable = '0'
      Result   : OUT std_logic  -- Inverted copy of the Data port
  );
END VITAL_INVIF0;
```

Inverter with active high Enable

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_INVIF1 IS
  GENERIC (
      ------------------------------------------------------------
      -- Timing Data
      ------------------------------------------------------------
      Tval             : TvalType       := VitalBUFFERdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data     : IN  std_logic; -- Input data to the gate
      Enable   : IN  std_logic; -- ~Data -> Result if Enable = '1'
      Result   : OUT std_logic  -- Inverted copy of the Data port
  );
END VITAL_INVIF1;
```

9.4 PULLUP, PULLDOWN

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_PULLUP IS
  PORT (
      Result   : OUT std_logic-- Pullup value
  );
END VITAL_PULLUP;
-----------------------------------------------------------------------------------------------
ENTITY VITAL_PULLDOWN IS
  PORT (
      Result   : OUT std_logic-- Pulldown value
  );
END VITAL_PULLDOWN;
```

9.5 MULTIPLEXOR

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_MUX IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalMUX4delay;-- default is set for a 4-input gate
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      Size             : Natural        := VitalGateSize2;-- Number of data ports = 2**Size
      InstancePath     : string         := ""                 -- instance pathname
  );
  PORT (
      Data         : IN  std_logic_vector(0 to (2**Size-1));  -- Input data to the mux
      DataSelect   : IN  std_logic_vector(0 to Size);         -- Data(DataSelect) -> Result
      Result       : OUT std_logic                            -- Output from MUX
  );
END VITAL_MUX;
```

9.6 DECODER

```
-----------------------------------------------------------------------------------------------
ENTITY VITAL_DECODER IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalDECODERdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;
      Size             : Natural        := VitalGateSize2;    -- Size of the DATA port
      InstancePath     : string     := ""                     -- instance pathname
  );
  PORT (
      Data     : IN  std_logic_vector(0 to (Size-1));     -- Input data to the mux
      Enable   : IN  std_logic;                           -- Result(...) = '0' if Enable = '0'
      Result   : OUT std_logic_vector(0 to (2**Size-1))   -- Result(Data) <= '1'
  );
END VITAL_DECODER;
```

9.7 D and SCAN Flip-Flops

DFF

```
-------------------------------------------------------------------------------------------------
ENTITY VITAL_DFF IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalDFFdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE; -- if enabled, Timing Checks performed
      XDetectionOn     : Boolean        := TRUE; -- if enabled, X's will be asserted
      XGenerationOn    : Boolean        := TRUE; -- if enabled and a Timing Violation occurs,
                                                 --  then the model will generate X's
      InstancePath     : string         := ""    -- instance pathname
  );
  PORT (
      DATA     : IN    std_logic; -- DATA INPUT
      CLK      : IN    std_logic; -- CLOCK
      Q        : INOUT std_logic; -- FLOP OUT
      Q_N      : OUT   std_logic  -- FLOP INVERTED OUT
  );

END VITAL_DFF;
```

DFF with Clear and Set

```
ENTITY VITAL_DFFCS IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalDFFCSdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE; -- if enabled, Timing Checks performed
      XDetectionOn     : Boolean        := TRUE; -- if enabled, X's will be asserted
      XGenerationOn    : Boolean        := TRUE; -- if enabled and a Timing Violation occurs,
                                                 --  then the model will generate X's
      InstancePath     : string         := ""    -- instance pathname
  );
  PORT (
      DATA     : IN    std_logic; -- DATA INPUT
      CLK      : IN    std_logic; -- CLOCK
      CLR      : IN    std_logic; -- ASYNCHRONOUS CLEAR
      SET      : IN    std_logic; -- ASYNCHRONOUS SET
      Q        : INOUT std_logic; -- FLOP OUT
      Q_N      : OUT   std_logic  -- FLOP INVERTED OUT
  );

END VITAL_DFFCS;
```

SCAN DFF with Clear and Set

```
-------------------------------------------------------------------------------------------------
ENTITY VITAL_SCANFF IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalSCANFFdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE;-- if enabled, Timing Checks performed
      XDetectionOn     : Boolean        := TRUE;-- if enabled, X's will be asserted
      XGenerationOn    : Boolean        := TRUE;-- if enabled and a Timing Violation occurs,
                                                --  then the model will generate X's
      InstancePath     : string         := ""   -- instance pathname
  );
  PORT (
      DATA     : IN    std_logic; -- DATA INPUT
      SCAN     : IN    std_logic; -- SCAN INPUT
      CLK      : IN    std_logic; -- CLOCK
      CLR      : IN    std_logic; -- ASYNCHRONOUS CLEAR
      SET      : IN    std_logic; -- ASYNCHRONOUS SET
      SCANSEL  : IN    std_logic; -- if (SCANSEL = '1') then input := DATA else input := SCAN;
      Q        : INOUT std_logic; -- FLOP OUT
      Q_N      : OUT   std_logic  -- FLOP INVERTED OUT
  );

END VITAL_SCANFF;
```

9.8 LATCH

```
-------------------------------------------------------------------------------------------------
ENTITY VITAL_LATCH IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalLATCHdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE; -- if enabled, Timing Checks performed
      XDetectionOn     : Boolean        := TRUE; -- if enabled, X's will be asserted
      XGenerationOn    : Boolean        := TRUE; -- if enabled and a Timing Violation occurs,
                                                 --  then the model will generate X's
      InstancePath     : string         := ""    -- instance pathname
  );
  PORT (
      DATA     : IN    std_logic; -- DATA INPUT
      CLK      : IN    std_logic; -- GATE
      Q        : INOUT std_logic; -- FLOP OUT
      Q_N      : OUT   std_logic  -- FLOP INVERTED OUT
  );

END VITAL_LATCH;
```

9.9 JKFF

```
------------------------------------------------------------------------------------------------
ENTITY VITAL_JKFF IS
  GENERIC (
      -------------------------------------------------------------
      -- Timing Data
      -------------------------------------------------------------
      Tval             : TvalType       := VitalJKFFdelay;
      Timemode         : TimemodeType   := t_typical;
      TimingChecksOn   : Boolean        := TRUE; -- if enabled, Timing Checks performed
      XDetectionOn     : Boolean        := TRUE; -- if enabled, X's will be asserted
      XGenerationOn    : Boolean        := TRUE; -- if enabled and a Timing Violation occurs,
                                                 --  then the model will generate X's
      InstancePath     : string         := ""    -- instance pathname
  );
  PORT (
      J            : IN    std_logic; -- J INPUT
      K            : IN    std_logic; -- K INPUT
      CLK          : IN    std_logic; -- GATE
      Q            : INOUT std_logic; -- FLOP OUT
      Q_N          : OUT   std_logic  -- FLOP INVERTED OUT
  );
END VITAL_JKFF;
```

10.0 Appendix A : Additional Recommended Guidelines

The following guidelines are not requirements under the VITAL modeling specification, but are provided for serious consideration in developing models in general.

10.1 Entity Naming Conventions

Certain simulators use the name of the entity 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).

Therefore, limit entity names to less than 28 characters.

10.2 Port Naming Conventions

There are few areas in which it is appropriate to restrict or suggest naming conventions. The following have been areas in which experience has shown that some constraint helps to provide interoperable models.

Active Low Ports: Append the suffix “_N” to the port name to designate the port as active low. (i.e. A port named I becomes I_N instead of IN and a port named DTACKN becomes DTACKN_N so that we can always identify the inversion by the “_N” rather than simply an N).

Naming in General: Experience favors compound names using upper and lower case characters to separate words rather than using underscores. Even though VHDL is case insensitive, few VHDL programmers use all lower case or all upper case characters during their development. Using mixed case is an effective tool for readability. (i.e. use DataBus instead of Data_Bus)

10.3 Generic Parameter Control Switches

In addition to the switch which controls the selection of minimum, typical or maximum timing parameters (TimeMode), each model should have a generic parameter named ***InstancePath*** which is used by the model to reflect the hierarchical pathname of the model’s instantiation in any error messages which the model should happen to issue.

```
-------------------------------------------------------------
-- Switches
-------------------------------------------------------------
XDetectionOn    : Boolean := TRUE;
TimingChecksOn : Boolean := TRUE;
XGenerationOn : Boolean := TRUE;
InstancePath    : string  := "" required, thread through 'pathname under 1992 VHDL
```

Other switches, such as XdetectionOn, TimingChecksOn and XGenerationOn may be used with a given modeling style as may be required. These additional switches are not a part of this interface standard, but are provided as suggestions on modeling style.

10.4 Naming Conventions for timing parameters

The following transition prefixes are encouraged in constructing names.

<prefix>_<InputPortName>_<OutputPortName>

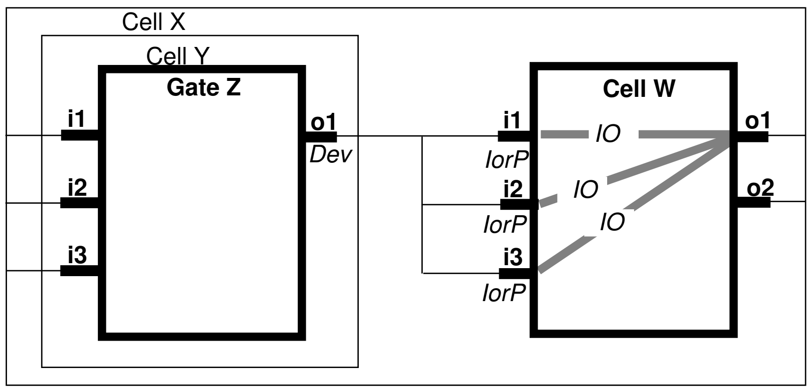
Table 2: Transition Dependencies

Input Port	Output Port	prefix
‘0‘	‘1‘	tplh
‘0‘	‘Z‘	tplz
‘1’	‘0’	tphl
‘1’	‘Z’	tphz
‘Z’	‘0’	tpzl
‘Z’	‘1’	tpzh
‘0’ -> ‘1’	‘1’	tpph
‘1‘ -> ‘0‘	‘1’	tpnh
‘0’ -> ‘1’	‘0’	tppl
‘1’ -> ‘0’	‘0’	tpnl
any -> ‘1’	‘1’	tprh
any -> ‘1’	‘0’	tprl
any -> ‘0’	‘1’	tpfh
any -> ‘0’	‘0’	tpfl

11.0 Appendix B: SDF Overview

The SDF timing model is an abstract representation which provides a framework for defining the specific elements required for library elements. The model allows these elements such as delay types, timing checks, constraints, and naming hierarchies to be defined in terms of the intrinsic data and interconnect relationships independent of specific tools or input languages.

The model is best understood in terms of a graphic:



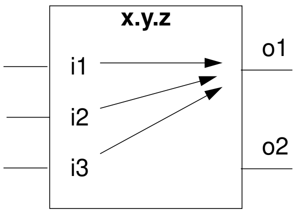
Abstract Timing Model

Dev = DEVICE delays (Intrinsic delays)

IorP = INTERCONNECT delays or PORT delays

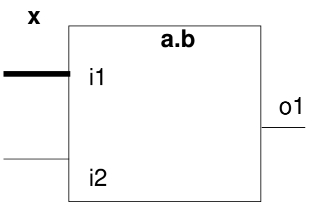
IO = IOPATH delays (Cell path delays)

Delay Types

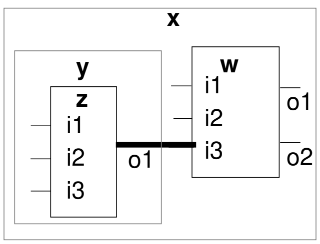


- **IOPATH** represents the delays on a legal path from an input port to an output port of a device. Each delay value is associated with a unique input port/output port pair.

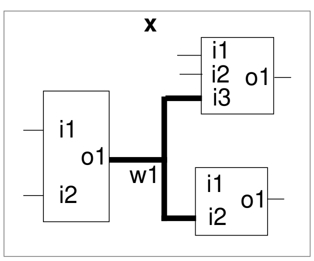
.



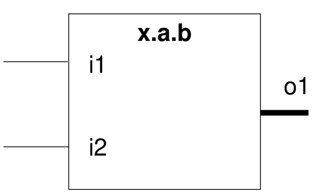
- The **PORT** represents the estimated interconnect signal delay values you can place in the input port, without having to specify a start point for the delay path.



- The **INTERCONNECT** represents actual or estimated delays in the wire paths between devices.

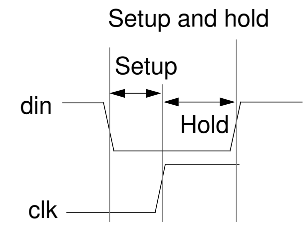


- The **NETDELAY** represents a delay for a complete net, where delays from all the source port(s) on the net to all destination port(s) have the same value. NETDELAY is a short form of INTERCONNECT delay.

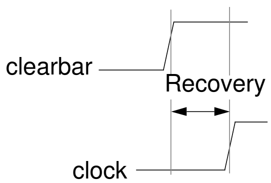


- The **DEVICE** keyword represents the intrinsic delay of a module or gate. Intrinsic delay is specific to the type of the object and so has the same value for every instance of that module or gate. Conceptually, this represents all path delays through the object, independent of loading or input slope. At the gate level the delay is associated with the output.

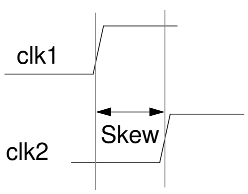
Timing Check or Constraints



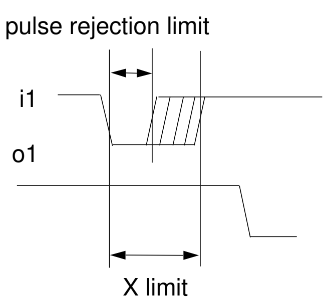
- The **SETUP, HOLD and SETUPHOLD** timing check represents limits on timing of a signal relative to a reference signal.
  The basic definition for setup and hold requirements is that the data signal must remain stable for a specified minimum time interval in order for a transition of the clock signal to store the data successfully. Setup time defines the minimum interval before the clock transition; hold time defines the minimum interval after the clock transition. Any change to the data signal within these intervals results in a timing violation. To shift the violation intervals with respect to the clock transition, either the setup time or hold time can be negative; however, their sum must always be greater than zero.



- The **RECOVERY** timing check is a constraint between a change in an asynchronous control signal and the next clock pulse, for example between clearbar and the clock for a flip-flop. If the clock signal violates the constraint, the output value is unknown—it can be the value set by the clearbar, or it can be the value clocked into the device.The **SKEW** timing check represents the limits for signal skew timing checks.



- A signal skew limit is the maximum allowable delay between two signals, which if exceeded causes devices to behave unreliably. For example, in a chain of devices such as counters that are connected to the same clock signal, the clock ideally changes at exactly the same time at all counters. However, there are some delay between the change at one device and the change at another. If this delay exceeds the signal skew limit, the data passed along the chain is unreliable. The skew limit between signals are identified as a design constraint by timing analysis tools. This information can be passed through the SDF file to layout tools to ensure that the physical design is laid out within these constraints.



Pulse rejection limit
This limit defines the minimum pulse width required for the pulse to pass through to the output; anything smaller does not affect the output.

- X limit
  This limit defines the minimum pulse width necessary to drive the output to a known state; anything smaller causes the output to enter the unknown (X) state or is rejected (if smaller than the pulse rejection limit). Note that the X limit must be greater than the pulse rejection limit to carry any significance.

Specify PATHPULSE in time units.

Example

```
(INSTANCE x)
(DELAY
    (PATHPULSE y.i1 y.o1 (13) (21))
)
```

The first value (13) is the pulse rejection limit and the second (21) is the X limit.

11.1 Appendix C: Detailed SDF Example

```
-- This line is ILLEGAL SDF. Lines starting with double dash -- will be comments hereafter.
-- ILLEGAL Constant names and array elements are used instead of actual values for illustration
(DELAYFILE
(SDFVERSION "1.0")
(DESIGN "VITAL Example")
(DATE "Tuesday, February 02, 1993 9:56 am")
(VENDOR "VHDL Technology Group")
(PROGRAM "hand coded")
(VERSION "v1.0")
(DIVIDER /)
(VOLTAGE 4.5:5.0:5.5)
(PROCESS "typical")
(TEMPERATURE 0:25:125)
(TIMESCALE 1 ps)
  (CELL
      (CELLTYPE "SCAN DFF with SET, CLR, QBAR derived from Q, ZOUT derived from Q,OEN")
      (INSTANCE /U21)
      (DELAY
          (ABSOLUTE(PORT D   (twl_D(t_min):twl_D(t_typ):twl_D(t_max))    -- conceptually
                             (twh_D(t_min):twh_D(t_typ):twh_D(t_max)))) -- conceptually
          (ABSOLUTE(PORT CLK (twl_CLK(t_min):twl_CLK(t_typ):twl_CLK(t_max))
                                (twh_CLK(t_min):twh_CLK(t_typ):twh_CLK(t_max))))
          (ABSOLUTE(PORT CLR (twl_CLR(t_min):twl_CLR(t_typ):twl_CLR(t_max))
                                (twh_CLR(t_min):twh_CLR(t_typ):twh_CLR(t_max))))
          (ABSOLUTE(PORT SET (twl_SET(t_min):twl_SET(t_typ):twl_SET(t_max))
                                (twh_SET(t_min):twh_SET(t_typ):twh_SET(t_max))))
          (ABSOLUTE(PORT SCAN (twl_SCAN(t_min):twl_SCAN(t_typ):twl_SCAN(t_max))
                                (twh_SCAN(t_min):twh_SCAN(t_typ):twh_SCAN(t_max))))
          (ABSOLUTE(PORT SCANSEL (twl_SCANSEL(t_min):twl_SCANSEL(t_typ):twl_SCANSEL(t_max))
                                (twh_SCANSEL(t_min):twh_SCANSEL(t_typ):twh_SCANSEL(t_max))))
          (ABSOLUTE(PORT Q   (twl_Q(t_min):twl_Q(t_typ):twl_Q(t_max))
                                        (twh_Q(t_min):twh_Q(t_typ):twh_Q(t_max))))
          (ABSOLUTE(PORT QBAR (twl_QBAR(t_min):twl_QBAR(t_typ):twl_QBAR(t_max))
                                        (twh_QBAR(t_min):twh_QBAR(t_typ):twh_QBAR(t_max))))
          (ABSOLUTE
              (COND    (CLR && SET && (posedge CLK))
                  (IOPATHCLKQ   (tplh_CLK_Q(t_min):tplh_CLK_Q(t_typ):tplh_CLK_Q(t_max))
                                (tphl_CLK_Q(t_min):tphl_CLK_Q(t_typ):tphl_CLK_Q(t_max)))))
          (ABSOLUTE
              (COND    (~CLR && SET)
                  (IOPATHCLRQ   (tphl_CLR_Q(t_min):tphl_CLR_Q(t_typ):tphl_CLR_Q(t_max)))))
          (ABSOLUTE
              (COND    (CLR && ~SET)
                  (IOPATHSETQ   (tplh_SET_Q(t_min):tplh_SET_Q(t_typ):tplh_SET_Q(t_max)))))
          (ABSOLUTE
              (COND    ((CLR == 'X') && ~SET && Q)
                  (IOPATHCLRQ   (tphl_CLR_Q(t_min):tphl_CLR_Q(t_typ):tphl_CLR_Q(t_max)))))
          (ABSOLUTE
              (COND    (CLR && (SET == 'X') && ~Q)
                  (IOPATHSETQ   (tplh_SET_Q(t_min):tplh_SET_Q(t_typ):tplh_SET_Q(t_max)))))
          (ABSOLUTE
              (COND    ((CLR == 'X') && (SET == 'X'))
                  (IOPATHSETQ   (tplh_SET_Q(t_min):tplh_SET_Q(t_typ):tplh_SET_Q(t_max)))))
          (ABSOLUTE(IOPATHQQBAR (tplh_Q_QBAR(t_min):tplh_Q_QBAR(t_typ):tplh_Q_QBAR(t_max))
                             (tphl_Q_QBAR(t_min):tphl_Q_QBAR(t_typ):tphl_Q_QBAR(t_max))))
          (TIMINGCHECK
              (COND    (CLR && SET && ~SCANSEL)
```

```
                    (SETUPHOLDD(posedge CLK)
                              ((tsetuph_D_CLK(t_min):tsetuph_D_CLK(t_typ):tsetuph_D_CLK(t_max))
                              (tsetupl_D_CLK(t_min):tsetupl_D_CLK(t_typ):tsetupl_D_CLK(t_max))))
                              ((tholdh_D_CLK(t_min):tholdh_D_CLK(t_typ):tholdh_D_CLK(t_max))
                              (tholdl_D_CLK(t_min):tholdl_D_CLK(t_typ):tholdl_D_CLK(t_max)))))
                    (COND    (CLR && SET && SCANSEL)
                         (SETUPHOLDSCAN(posedge CLK)
                         (tsetuph_SCAN_CLK(t_min):tsetuph_SCAN_CLK(t_typ):tsetuph_SCAN_CLK(t_max))
                          (tsetupl_SCAN_CLK(t_min):tsetupl_SCAN_CLK(t_typ):tsetupl_SCAN_CLK(t_max))))
                          ((tholdh_SCAN_CLK(t_min):tholdh_SCAN_CLK(t_typ):tholdh_SCAN_CLK(t_max))
                          (tholdl_SCAN_CLK(t_min):tholdl_SCAN_CLK(t_typ):tholdl_SCAN_CLK(t_max)))))
                    (COND    (CLR && SET)
                         (SETUPHOLDSCANSEL(posedge CLK)
                          ((tsetuph_SCANSEL_CLK(t_min):
                             tsetuph_SCANSEL_CLK(t_typ):tsetuph_SCANSEL_CLK(t_max))
                          (tsetupl_SCANSEL_CLK(t_min):
                             tsetupl_SCANSEL_CLK(t_typ):tsetupl_SCANSEL_CLK(t_max))))
                          ((tholdh_SCANSEL_CLK(t_min):tholdh_SCANSEL_CLK(t_typ):tholdh_SCANSEL_CLK(t_max))
                     (tholdl_SCANSEL_CLK(t_min):tholdl_SCANSEL_CLK(t_typ):tholdl_SCANSEL_CLK(t_max)))))
                    (COND    (CLR && SET)
                         (RECOVERYCLR(posedge CLK)
                     (trecoveryh_CLR_CLK(t_min):trecoveryh_CLR_CLK(t_typ):trecoveryh_CLR_CLK(t_max))
                     (trecoveryl_CLR_CLK(t_min):trecoveryl_CLR_CLK(t_typ):trecoveryl_CLR_CLK(t_max))))
                    (COND    (CLR && SET)
                         (HOLD    CLR  (posedge CLK)
                     (trecoveryh_CLR_CLK(t_min):trecoveryh_CLR_CLK(t_typ):trecoveryh_CLR_CLK(t_max))
                         (trecoveryl_CLR_CLK(t_min):trecoveryl_CLR_CLK(t_typ):trecoveryl_CLR_CLK(t_max))))
                    (COND    (CLR && SET)
                         (RECOVERYSET(posedge CLK)
                     (trecoveryh_SET_CLK(t_min):trecoveryh_SET_CLK(t_typ):trecoveryh_SET_CLK(t_max))
                     (trecoveryl_SET_CLK(t_min):trecoveryl_SET_CLK(t_typ):trecoveryl_SET_CLK(t_max))))
                    (COND    (CLR && SET)
                         (HOLD    SET  (posedge CLK)
                     (trecoveryh_SET_CLK(t_min):trecoveryh_SET_CLK(t_typ):trecoveryh_SET_CLK(t_max))
                     (trecoveryl_SET_CLK(t_min):trecoveryl_SET_CLK(t_typ):trecoveryl_SET_CLK(t_max))))
                    (WIDTH(negedge CLK) (tpwl_CLK(t_min):tpwl_CLK(t_typ):tpwl_CLK(t_max)))
                    (WIDTH(posedge CLK) (tpwh_CLK(t_min):tpwh_CLK(t_typ):tpwh_CLK(t_max)))
               )
)))
```

12.0 Appendix D: Example VITAL Model (Scan Flip-Flop with SET, CLR)

> **Each timing parameter has a unique integer index value**

```
-- --------------------------------------------------------------------
--                                     Developed by...
--                                     The VHDL Technology Group
--   File name :  Tvalpkg.vhd
--   Title     :  Package to accompany entity SCANFF
--   Library   :  VITAL
--   Author(s) :  W. Billowitch
-- --------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- --------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V1.000   |  wdb   |  11/20/92  | Original Code
-- --------------------------------------------------------------------
LIBRARY Std_DevelopersKit;
USE Std_DevelopersKit.Std_Timing.all; -- used to test the implementation
-- ---------------------------------------------------------------------

omatically generated from an SDF file.
name the package Tvalpkg_<entityname>.
----------------------------------------
PACKAGE Tvalpkg_scanff IS
       CONSTANT tplh_CLK_Q     : INTEGER := 1; -- Propagation delay from CLK to Q='1'
       CONSTANT tphl_CLK_Q     : INTEGER := 2; -- Propagation delay from CLK to Q='0'
       CONSTANT tphl_CLR_Q     : INTEGER := 3; -- Propagation delay from CLR to Q='0'
       CONSTANT tplh_SET_Q     : INTEGER := 4; -- Propagation delay from SET to Q='1'
       CONSTANT tplh_Q_Q_N     : INTEGER := 5; -- Propagation delay from Q to QBAR='1'
       CONSTANT tphl_Q_Q_N     : INTEGER := 6; -- Propagation delay from Q to QBAR='0'
 -- Timing Checks
       CONSTANT tsetuph_D_CLK       : INTEGER := 7; -- Setup check on D='1' w.r.t. CLK
       CONSTANT tsetupl_D_CLK       : INTEGER := 8; -- Setup Check on D='0' w.r.t. CLK
       CONSTANT tholdh_D_CLK        : INTEGER := 9; -- Hold Check on D='1' w.r.t. CLK
       CONSTANT tholdl_D_CLK        : INTEGER := 10; -- Hold Check on D='0' w.r.t. CLK
       CONSTANT tsetuph_SCAN_CLK    : INTEGER := 11; -- Setup Check on SCAN='1' w.r.t. CLK
       CONSTANT tsetupl_SCAN_CLK    : INTEGER := 12; -- Setup Check on SCAN='0' w.r.t. CLK
       CONSTANT tholdh_SCAN_CLK     : INTEGER := 13; -- Hold Check on SCAN='1' w.r.t. CLK
       CONSTANT tholdl_SCAN_CLK     : INTEGER := 14; -- Hold Check on SCAN='0' w.r.t. CLK
       CONSTANT tsetuph_SCANSEL_CLK : INTEGER := 15; -- Setup Check on SCANSEL='1' w.r.t. CLK
       CONSTANT tsetupl_SCANSEL_CLK : INTEGER := 16; -- Setup Check on SCANSEL='0' w.r.t. CLK
       CONSTANT tholdh_SCANSEL_CLK  : INTEGER := 17; -- Hold Check on SCANSEL='1' w.r.t. CLK
       CONSTANT tholdl_SCANSEL_CLK  : INTEGER := 18; -- Hold Check on SCANSEL='0' w.r.t. CLK
       CONSTANT tpwl_CLK            : INTEGER := 19; -- Pulse Width (low) Check on CLK
       CONSTANT tpwh_CLK            : INTEGER := 20; -- Pulse Width (high) Check on CLK
       CONSTANT trecoveryh_CLR_CLK  : INTEGER := 21; -- Recovery Check on CLR='1' w.r.t. CLK
       CONSTANT trecoveryl_CLR_CLK  : INTEGER := 22; -- Recovery Check on CLR='0' w.r.t. CLK
       CONSTANT tholdh_CLR_CLK      : INTEGER := 23; -- Hold Check on CLR='1' w.r.t. CLK
       CONSTANT tholdl_CLR_CLK      : INTEGER := 24; -- Hold Check on CLR='0' w.r.t. CLK
       CONSTANT trecoveryh_SET_CLK  : INTEGER := 25; -- Recovery Check on SET='1' w.r.t. CLK
       CONSTANT trecoveryl_SET_CLK  : INTEGER := 26; -- Recovery Check on SET='0' w.r.t. CLK
       CONSTANT tholdh_SET_CLK      : INTEGER := 27; -- Hold Check on SET='1' w.r.t. CLK
       CONSTANT tholdl_SET_CLK      : INTEGER := 28; -- Hold Check on SET='0' w.r.t. CLK
       -- -------------------------------------------------------------------------------
```

```
        -- Deferred Constant
        --      Note:  By using deferred constants, you can update the VALUES of the constants
        --             and re-compile just the package body in order for the simulation to take
        --             on the new values. This technique is preferable since it does not
        --             require the re-compilation of the package and associated model and
        --             configurations.
        -- ----------------------------------------------------------------------------------
        CONSTANT Tdefault_scanff : TvalType (1 to 28); --
END Tvalpkg_scanff;

PACKAGE BODY Tvalpkg_scanff IS
        -- -------------------------------------------
        -- _DEFAULT
        -- Note:There shall be one constant with the prefix "Tdefault_" to be used in the
        --             development of the associated model.
        -- ------------------------------------------------------------------------------
                t_scanff : TvalType(1 to 28) := (
                            => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from CLK to Q='1'
         tphl_CLK_Q         => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from CLK to Q='0'
         tphl_CLR_Q         => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from CLR to Q='0'
         tplh_SET_Q         => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from SET to Q='1'
         tplh_Q_Q_N         => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from Q to QBAR='1'
         tphl_Q_Q_N         => ( 2 ns, 3 ns, 4 ns ), -- Propagation delay from Q to QBAR='0'
         tsetuph_D_CLK      => ( 2 ns, 3 ns, 4 ns ), -- Setup check on D='1' w.r.t. CLK
         tsetupl_D_CLK      => ( 2 ns, 3 ns, 4 ns ), -- Setup CHECK on D='0' w.r.t. CLK
         tholdh_D_CLK       => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on D='1' w.r.t. CLK
         tholdl_D_CLK       => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on D='0' w.r.t. CLK
         tsetuph_SCAN_CLK   => ( 2 ns, 3 ns, 4 ns ), -- Setup CHECK on SCAN='1' w.r.t. CLK
         tsetupl_SCAN_CLK   => ( 2 ns, 3 ns, 4 ns ), -- Setup CHECK on SCAN='0' w.r.t. CLK
         tholdh_SCAN_CLK    => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on SCAN='1' w.r.t. CLK
         tholdl_SCAN_CLK    => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on SCAN='0' w.r.t. CLK
         tsetuph_SCANSEL_CLK => ( 2 ns, 3 ns, 4 ns ), -- Setup CHECK on SCANSEL='1' w.r.t. CLK
         tsetupl_SCANSEL_CLK => ( 2 ns, 3 ns, 4 ns ), -- Setup CHECK on SCANSEL='0' w.r.t. CLK
         tholdh_SCANSEL_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on SCANSEL='1' w.r.t. CLK
         tholdl_SCANSEL_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on SCANSEL='0' w.r.t. CLK
         tpwl_CLK           => ( 2 ns, 3 ns, 4 ns ), -- Pulse Width (low) CHECK on CLK
         tpwh_CLK           => ( 2 ns, 3 ns, 4 ns ), -- Pulse Width (high) CHECK on CLK
         trecoveryh_CLR_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Recovery CHECK on CLR='1' w.r.t. CLK
         trecoveryl_CLR_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Recovery CHECK on CLR='0' w.r.t. CLK
         tholdh_CLR_CLK     => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on CLR='1' w.r.t. CLK
         tholdl_CLR_CLK     => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on CLR='0' w.r.t. CLK
         trecoveryh_SET_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Recovery CHECK on SET='1' w.r.t. CLK
         trecoveryl_SET_CLK  => ( 2 ns, 3 ns, 4 ns ), -- Recovery CHECK on SET='0' w.r.t. CLK
         tholdh_SET_CLK     => ( 2 ns, 3 ns, 4 ns ), -- Hold CHECK on SET='1' w.r.t. CLK
         tholdl_SET_CLK     => ( 2 ns, 3 ns, 4 ns ) -- Hold CHECK on SET='0' w.r.t. CLK
        );
END Tvalpkg_scanff;
```

Deferred Constant values provide default timing values for the model

Package name is intuitive and unique to the model

```
-- --------------------------------------------------------------------
--                                                  Developed by...
--                                                  The VHDL Technology Group
--   File name :  scannff.vhd
--   Title     :  D FLIP-FLOP WITH SYNCH SCAN + ASYNCHRONOUS CLEAR AND SET
--   Library   :  VITAL
--   Author(s) :  W. Billowitch
-- --------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- --------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V1.000   |  wdb   |  11/03/92  | Original Code
-- --------------------------------------------------------------------
LIBRARY IEEE;                 USE IEEE.Std_logic_1164.all;
LIBRARY Std_DevelopersKit;    USE Std_DevelopersKit.Std_Timing.all;
LIBRARY VITAL;                USE VITAL.Tvalpkg_scanff.all;
----------------------------------------------------------------------------------------
-- ENTITY declaration
```

Even the most complex timing models can be represented by the same interface declaration!

```
--------------------------------------------
ENTITY SCANFF IS
  GENERIC (
        -------------------------------------------------------------
        -- Timing Data   Note:No matter how many parameters there are,
        --                    the generic interface is never more complex.
        -------------------------------------------------------------
        Tval           : TvalType := Tdefault_scanff;

        -------------------------------------------------------------
        -- Switches
        -------------------------------------------------------------
        Timemode       : TimemodeType    := t_typical;
        XDetectionOn   : Boolean:= TRUE;
        TimingChecksOn : Boolean:= TRUE;
        XGenerationOn : Boolean := TRUE;
        InstancePath   : string := "" -- thread through 'pathname under 1992 VHDL
        );
  PORT    (
        D       : IN  std_logic; -- DATA
        CLK     : IN  std_logic; -- CLOCK
        CLR     : IN  std_logic; -- CLEAR
        SET     : IN  std_logic; -- SET
        SCAN    : IN  std_logic; -- SCAN INPUT
        SCANSEL : IN  std_logic; -- SCAN ENABLE
        Q       : INOUT std_logic; -- D FLOP OUT
        Q_N     : OUT std_logic  -- D FLOP INVERTED OUT
        );
END SCANFF;
```

```
----------------------------------------------------------------------
-- ARCHITECTURAL declaration
----------------------------------------------------------------------
ARCHITECTURE BEHAVIORAL OF SCANFF IS
BEGIN
    --------------------------------------------------------
    -- FUNCTIONALITY
    --------------------------------------------------------
macrocell : PROCESS ( CLK, CLR, SET )
       VARIABLE CLK_X01     : std_ulogic;
       VARIABLE CLR_X01     : std_ulogic;
       VARIABLE SET_X01     : std_ulogic;
       VARIABLE SCAN_X01    : std_ulogic;
       VARIABLE SCANSEL_X01 : std_ulogic;
       VARIABLE D_X01       : std_ulogic;
       VARIABLE inputmux    : std_ulogic;
       -- these two variable exist solely to provide
       -- non-pessimistic X propagation
       VARIABLE Qlastvalue  : std_ulogic;
       VARIABLE Qlastevent  : time;
BEGIN
       CLK_X01     := To_X01(CLK);
       CLR_X01     := To_X01(CLR);
       SET_X01     := To_X01(SET);
       SCAN_X01    := To_X01(SCAN);
       SCANSEL_X01 := To_X01(SCANSEL);
       D_X01       := To_X01(D);
       -- ---------------------------------------------------------------------
       -- DETAILED FUNCTIONAL DESCRIPTION WITH FULL X DETECTION
       -- ---------------------------------------------------------------------
       IF (CLR_X01 = '1') and (SET_X01 = '1') then
           inputmux := (SCANSEL and SCAN ) or ((not SCANSEL ) and D );
           IF rising_edge(CLK) then
              Qlastevent := CalcDelay (inputmux, Qlastvalue,
                     Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode));
              Qlastvalue := inputmux;
              IF XDetectionOn THEN
                  assert (D  /= 'X') or (SCANSEL  = '1')
                  report "X observed on D port for instance" & InstancePath severity note;
                  assert (SCAN  /= 'X') or (SCANSEL  = '0')
                  report "X observed on SCAN port for instance" & InstancePath severity note;
              END IF;
           ELSIF (CLK_X01 = 'X') then
              IF (Qlastvalue /= inputmux) THEN
                  Qlastevent := CalcDelay ('X', Qlastvalue,
                     Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode));
                  Qlastvalue := 'X';
              END IF;
              IF XDetectionOn THEN
                  assert false report "X observed on CLK port for instance" &
                      InstancePath severity ERROR;
              END IF;
           END IF;
```

```
        ELSIF (CLR_X01 = '0') and (SET_X01 = '1') then
            Qlastevent := CalcDelay('1', '-', Tval(tplh_SET_Q)(TimeMode));
            Qlastvalue := '1';
        ELSIF (CLR_X01 = '1') and (SET_X01 = '0') then
            Qlastevent := CalcDelay('0', '-', Tval(tphl_CLR_Q)(TimeMode));
            Qlastvalue := '0';
        ELSIF (CLR_X01 = 'X') and (SET_X01 = '0') then
            -- only a problem if Q = '1'
            IF (Qlastvalue = '1') THEN
               Qlastevent := CalcDelay('X', '-', Tval(tphl_CLR_Q)(TimeMode));
               Qlastvalue := 'X';
               assert (XDetectionOn /= TRUE)
               report "X observed on CLR port for instance " & InstancePath
               severity ERROR;
            END IF;
        ELSIF (CLR_X01 = '1') and (SET_X01 = 'X') then
            -- only a problem if Q = '0'
            IF (Qlastvalue = '0') THEN
               Qlastevent := CalcDelay('X', '-', Tval(tplh_SET_Q)(TimeMode));
               Qlastvalue := 'X';
               assert (XDetectionOn /= TRUE)
               report "X observed on SET port for instance " & InstancePath
               severity ERROR;
            END IF;
        ELSE
            Qlastevent := CalcDelay('X', '-', Tval(tphl_CLR_Q)(TimeMode));
            Qlastvalue := 'X';
            assert (XDetectionOn /= TRUE)
            report "X observed on SET and CLR ports for instance " & InstancePath
            severity ERROR;
        END IF;
        Q <= Qlastvalue after Qlastevent;
END PROCESS macrocell;


-- --------------------------------------------
-- Q_N derived from the Q
-- --------------------------------------------
        Q_N  <= (not Q ) after CalcDelay((not Q ), '-', Tval(tplh_Q_Q_N)(TimeMode), Tval(t-
phl_Q_Q_N)(TimeMode));
```

```
-- --------------------------------------------------------------------------
-- TIMING CHECKS
-- --------------------------------------------------------------------------
timchk: IF (TimingChecksOn) GENERATE
checkit : PROCESS ( D, CLK, CLR, SET, SCAN, SCANSEL)
       VARIABLE CLK_X01     : std_ulogic;
       VARIABLE CLR_X01     : std_ulogic;
       VARIABLE SET_X01     : std_ulogic;
       VARIABLE SCAN_X01    : std_ulogic;
       VARIABLE SCANSEL_X01 : std_ulogic;
       VARIABLE D_X01       : std_ulogic;
       VARIABLE timemark    : time;
BEGIN
       CLK_X01     := To_X01(CLK);
       CLR_X01     := To_X01(CLR);
       SET_X01     := To_X01(SET);
       SCAN_X01    := To_X01(SCAN);
       SCANSEL_X01 := To_X01(SCANSEL);
       D_X01       := To_X01(D);
       -----------------------------------------------------------------
       -- DATA setup & hold checks
       -- Note: most compact coding style shown (Verilog-like)
       -----------------------------------------------------------------
       if TimingViolation (D,"D",CLK,"CLK",
              Tval(tsetuph_D_CLK)(TimeMode), Tval(tsetupl_D_CLK)(TimeMode),
              Tval(tholdh_D_CLK)(TimeMode) , Tval(tholdl_D_CLK)(TimeMode),
              boolean'((CLK_X01 = '1') and (CLR_X01 /= '0') and
              (SET_X01 /= '0') and (SCANSEL_X01 /= '1')),
              InstancePath)
       and XGenerationOn then
              -- Corrupt the output during this clock cycle
              Q <= TRANSPORT 'X' after CalcDelay ( 'X', '-',
                   Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode) );
       else
              Q <= TRANSPORT 'Z';
       end if;

       ------------------------------------------------------------------
       -- SCAN setup & hold checks
       -- Note: more verbose, but MUCH better understood coding style
       ------------------------------------------------------------------
       if TimingViolation (
              testport => SCAN ,
              testportname => "SCAN",
              refport   => CLK ,
              refportname  => "CLK",
              t_setup_hi   => Tval(tsetuph_SCAN_CLK)(TimeMode),
              t_setup_lo   => Tval(tsetupl_SCAN_CLK)(TimeMode),
              t_hold_hi=> Tval(tholdh_SCAN_CLK)(TimeMode),
              t_hold_lo=> Tval(tholdl_SCAN_CLK)(TimeMode),
              condition=> boolean'((CLK_X01  = '1')  and
                     (CLR_X01  /= '0')  and
                     (SET_X01  /= '0')  and
                     (SCANSEL_X01  /= '0')),
```

If you don't want the model to perform timing violation checking. Turn it off!

```
        HeaderMsg=> InstancePath)
and XGenerationOn then
         -- Corrupt the output during this clock cycle
        Q          <= TRANSPORT 'X' after CalcDelay ( 'X', '-',
                       Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode) );
else
        Q <= TRANSPORT 'Z';
end if;
-- SCANSEL setup & hold checks
if TimingViolation (
        testport => SCANSEL ,
        testportname => "SCANSEL",
        refport   => CLK ,
        refportname  => "CLK",
        t_setup_hi   => Tval(tsetuph_SCANSEL_CLK)(TimeMode),
        t_setup_lo   => Tval(tsetupl_SCANSEL_CLK)(TimeMode),
        t_hold_hi=> Tval(tholdh_SCANSEL_CLK)(TimeMode),
        t_hold_lo=> Tval(tholdl_SCANSEL_CLK)(TimeMode),
        condition=> boolean'((CLK_X01  = '1')  and
                        (CLR_X01  /= '0')  and
                        (SET_X01  /= '0')),
        HeaderMsg=> InstancePath)
and XGenerationOn then
 -- Corrupt the output during this clock cycle
         Q         <= TRANSPORT 'X' after CalcDelay ( 'X', '-',
               Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode) );
else
        Q <= TRANSPORT 'Z';
end if;
-- CLR recovery check
IF TimingViolation (
        testport => CLR ,
        testportname => "CLR",
        refport   => CLK ,
        refportname  => "CLK",
        t_setup_hi => Tval(trecoveryh_CLR_CLK)(TimeMode),
        t_setup_lo => Tval(trecoveryl_CLR_CLK)(TimeMode),
        t_hold_hi=> Tval(tholdh_CLR_CLK)(TimeMode),
        t_hold_lo=> Tval(tholdl_CLR_CLK)(TimeMode),
        condition=> boolean'((CLR_X01  = '1')  and
                        (SET_X01  /= '0')),
        HeaderMsg=> InstancePath)
and XGenerationOn then
         -- Corrupt the output during this clock cycle
         Q         <= TRANSPORT 'X' after CalcDelay ( 'X','-',
               Tval(tplh_CLK_Q)(TimeMode), Tval(tphl_CLK_Q)(TimeMode) );
else
        Q <= TRANSPORT 'Z';
end if;

-- SET recovery check
IF TimingViolation (
        testport => SET ,
        testportname => "SET",
```

```
                refport   => CLK ,
                refportname  => "CLK",
                t_release_hi => Tval(trecoveryh_SET_CLK)(TimeMode),
                t_release_lo => Tval(trecoveryl_SET_CLK)(TimeMode),
                t_hold_hi=> Tval(tholdh_SET_CLK)(TimeMode),
                t_hold_lo=> Tval(tholdl_SET_CLK)(TimeMode),
                condition=> boolean'((CLR_X01  /= '0')  and
                               (SET_X01  = '1')),
                HeaderMsg=> InstancePath)
        and XGenerationOn then
                 -- Corrupt the output during this clock cycle
                 Q         <= TRANSPORT 'X' after
                       CalcDelay ( 'X', '-', Tval(tplh_CLK_Q)(TimeMode),
                                              Tval(tphl_CLK_Q)(TimeMode) );
        else
                Q <= TRANSPORT 'Z';
        end if;

        PulseCheck ( testport => CLK ,
                 testportname => "CLK",
                 t_pulse_hi   => Tval(tpwh_CLK)(TimeMode),
                 t_pulse_lo   => Tval(tpwl_CLK)(TimeMode),
                 timemark => timemark,
                 HeaderMsg=> InstancePath);
         END PROCESS checkit;
    END GENERATE;
END BEHAVIORAL;
```

13.0 Appendix E: VITAL Wire Delay Model Specification

Background: In general, there are two techniques possible for handling wire delays. The first is to incorporate the wire delay as a pre-delay or post-delay on an input or output ports respectively of the model. The other is to treat the wire delay as a separate component.

While these techniques work for simple topologies, they fail to provide an adequate solution for more extensive interconnect topologies. In addition, models which use these techniques can be quite inefficient due to the increased number of process statements within the model.

As an illustration, imagine a chain of buffers, each with an input and output delay process in addition to the buffer's behavioral process. In this case, we have three process statements per model, where one would do just fine. This type of inefficiency cannot be tolerated in high gate count designs. Therefore, in order to address the simultaneous requirements for efficiency and timing accuracy, the following VITAL wire model specification will be applied

VITAL Guideline: The VITAL wire delay specification calls for the instantiation of an independent wire delay model.

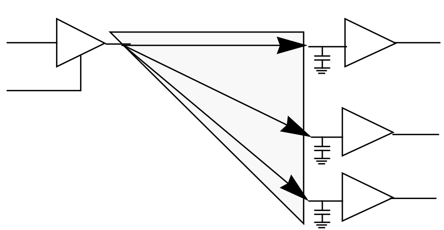
- When the need arises to model the delays associated with a wire delay, the user (or preferably a netlister tool) shall create and insert a model of a wire where necessary. In essence, do not encumber the model with the tasks of modeling the interconnection topology. Modeling the interconnection topology is the task of an interconnect model not a VITAL model.
- The advantages of a separate net model are the following:
- Lower process count for the typical interconnection topologies.
- Significantly simplified modeling methodology
- Easy separation of net models from functional models

Tool Impact: By following this methodology, tools can be written which generate wire-model-free circuits as well as wire-model-inclusive circuits at the throw of a switch. Such a tool could therefore allow the user to generate a wire-free netlist for a foundry while simulating the netlist with wire models included using their own workstation.

Further details of the wire delay modeling techniques can be found in the Appendix.

13.1 Simple Uni-Directional Single Driver-Multiple Receiver topology

Timing impact: In this case, all of the lumped capacitive loads are reflected back to the single driver. Each of the interconnection paths may have a unique path delay.

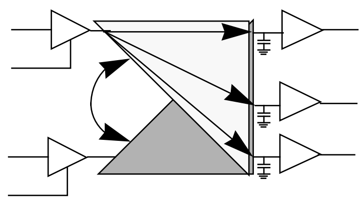


The wire model would consist of three assignments, each with their own delay parameters. The delay parameters may be state dependent.

SDF Impact: In this case, all of the transmission delay can be reflected to the inputs and the SDF INTERCONNECT syntax will suffice.

13.2 Multiple Driver-Multiple Receiver

Timing Impact: In this case, two models are created, one which is driven by the top driver, the other by the bottom driver. The loads being driven are shared between the drivers.



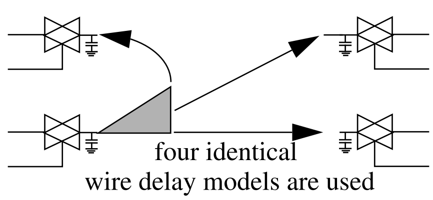
Wire delays and loading dependent delays are modeled on a pin-to-pin basis. Signal resolution takes place at the junction of the receivers.

Advanced Modeling: Furthermore, the two models can be interconnected via a signal to handle dynamic loading effect. One wire model can inform the other that its driver is no longer driving, thereby shifting the full load responsibility to the remaining drivers dynamically.

VITAL: Two classes of wire models are defined, one with and one without load sharing capability.

13.3 Multiple Bidirectional Driver-Multiple Bidirectional Receiver

Timing Impact: There are a number of complex timing situations which require advanced modeling of the interconnect delay. In particular, multiply driven bidirectional busses are one such requirement. In modeling bidirectional buses, a VITAL compliant implementation would decompose the bus topology into multiple uni-directional drivers much in the same fashion as modeling any other type of wire delay. One wire delay model will be instantiated per driver.



Wire delays and loading dependent delays are then modeled on a pin-to-pin basis. In the example shown, four identical wire delay models will be used each being driven by its respective driver. Signal resolution will then occur on the receiver pin. Since the 1164 standard offers a reflexive resolution function, each receiver will find the same value on its port as if the interconnection had occurred at a single point in the circuit.

One again, if sophisticated timing is required, the wire models can include a broadcast signal which informs the other wire delay models of the number of active drivers on the network. This would allow dynamic load dependent delay modeling with a minimum of overhead.

13.4 Wire Delay Model Implementation Example

```
-- --------------------------------------------------------------------
--                                          Developed by...
--                                          The VHDL Technology Group
--   File name :  wiremodl.vhd
--   Title     :  WIRE DELAY MODEL
--   Library   :  VITAL
--   Author(s) :  W. Billowitch
-- --------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- --------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V1.000   |  wdb   |   3/20/93  | Original Code
-- --------------------------------------------------------------------
LIBRARY IEEE;                USE IEEE.Std_logic_1164.all;
LIBRARY Std_DevelopersKit;   USE Std_DevelopersKit.Std_Timing.all;
LIBRARY VITAL;               USE VITAL.Tvalpkg_primitives.all;
-----------------------------------------------------------------------
-- ENTITY declaration
-----------------------------------------------------------------------
ENTITY WIREDELAY IS
  GENERIC (
       Tval                       : TvalType     := VitalWireDelay;
       Timemode                   : TimemodeType := t_typical;
       Size                       : INTEGER      := 2;   -- indicates the fanout count
       TimingChecksOn: Boolean       := TRUE;            -- Turns on HAZARD Detection
       XGenerationOn  : Boolean      := TRUE;            -- If enabled, detected HAZARDS
                                                         -- will generate X's
       InstancePath   : string       := ""  );
  PORT (
       INPUT          : IN std_logic;
       OUTPUT         : OUT std_logic_vector (0 to (Size-1))  );
END WIREDELAY;
ARCHITECTURE behavioral OF WIREDELAY IS
BEGIN
   primitive: PROCESS (INPUT)
   BEGIN
       for i in 0 to (Size-1) loop
              OUTPUT(i) <= TRANSPORT INPUT after
                     CalcDelay (INPUT, INPUT'last_value,
                            Tval(i  )(TimeMode), Tval(i+1)(TimeMode),
                            Tval(i+2)(TimeMode), Tval(i+3)(TimeMode),
                            Tval(i+4)(TimeMode), Tval(i+5)(TimeMode));
              end loop;
   END PROCESS;
   hazard_check: If (TimingChecksOn) GENERATE
       -- NOTE: HAZARD detection and XGeneration not implemented in this release.
   END GENERATE;
END behavioral;
```

14.0 Appendix F: Vital Primitives

14.1 Vital Primitives Timing Package

```
------------------------------------------------------------------------
--                          Developed by...
--                          The VHDL Technology Group
--
--   File name :  package.vhd
--   Title     :  Timing Value Package for all VITAL Primitives
--   Library   :  vital.lib
--   Notes     :
--   Author(s) :  W. Billowitch
-- ----------------------------------------------------------------------
--   MODIFICATION HISTORY :
-- ----------------------------------------------------------------------
--   version no:| author:|  mod. date:| changes made:
--     V1.000   |  wdb   |  03/24/93  | Original Code
-- ----------------------------------------------------------------------
LIBRARY Std_DevelopersKit;
USE    Std_DevelopersKit.Std_Timing.all;

PACKAGE Tvalpkg_primitives IS

    ---------------------------------------------------------------
    -- VitalGateSize: This establishes the number of inputs to the logical primitive.
    ---------------------------------------------------------------
    CONSTANT VitalGateSize1 : INTEGER := 1; -- commonly used sizes
    CONSTANT VitalGateSize2 : INTEGER := 2; -- commonly used sizes
    CONSTANT VitalGateSize4 : INTEGER := 4; -- commonly used sizes
    CONSTANT VitalGateSize6 : INTEGER := 6; -- commonly used sizes
    CONSTANT VitalGateSize8 : INTEGER := 8; -- commonly used sizes

    ---------------------------------------------------------------
    -- There are two parameters for every input to the logical primitives
    -- The convention is to allow the even numbered indicies
    -- contain the low to high transitions, and the odd numbered
    -- indicies handle the high to low transitions.
    ---------------------------------------------------------------
    CONSTANT tplh_0 : INTEGER := 0;    CONSTANT tphl_0 : INTEGER := 1;
    CONSTANT tplh_1 : INTEGER := 2;    CONSTANT tphl_1 : INTEGER := 3;
    CONSTANT tplh_2 : INTEGER := 4;    CONSTANT tphl_2 : INTEGER := 5;
    CONSTANT tplh_3 : INTEGER := 6;    CONSTANT tphl_3 : INTEGER := 7;
    CONSTANT tplh_4 : INTEGER := 8;    CONSTANT tphl_4 : INTEGER := 9;
    CONSTANT tplh_5 : INTEGER := 10;   CONSTANT tphl_5 : INTEGER := 11;
    CONSTANT tplh_6 : INTEGER := 12;   CONSTANT tphl_6 : INTEGER := 13;
    CONSTANT tplh_7 : INTEGER := 14;   CONSTANT tphl_7 : INTEGER := 15;

    CONSTANT VitalGateDelay1 : TvalType(tplh_0 to tphl_0);
    CONSTANT VitalGateDelay2 : TvalType(tplh_0 to tphl_1);
    CONSTANT VitalGateDelay4 : TvalType(tplh_0 to tphl_3);
    CONSTANT VitalGateDelay6 : TvalType(tplh_0 to tphl_5);
    CONSTANT VitalGateDelay8 : TvalType(tplh_0 to tphl_7);

    ---------------------------------------------------------------
    -- Three State Buffers and Inverters
    ---------------------------------------------------------------
    CONSTANT tplh_Data_Result   : INTEGER := 0;
```

```
CONSTANT tphl_Data_Result   : INTEGER := 1;
CONSTANT tplz_Enable_Result : INTEGER := 2;
CONSTANT tphz_Enable_Result : INTEGER := 3;
CONSTANT tpzl_Enable_Result : INTEGER := 4;
CONSTANT tpzh_Enable_Result : INTEGER := 5;

CONSTANT VitalBUFFERdelay : TvalType(0 to 5);

---------------------------------------------------------------
-- MUX
---------------------------------------------------------------
CONSTANT tplh_DataSelect3_Result : INTEGER :=  0;
CONSTANT tphl_DataSelect3_Result : INTEGER :=  1;
CONSTANT tplh_DataSelect2_Result : INTEGER :=  2;
CONSTANT tphl_DataSelect2_Result : INTEGER :=  3;
CONSTANT tplh_DataSelect1_Result : INTEGER :=  4;
CONSTANT tphl_DataSelect1_Result : INTEGER :=  5;
CONSTANT tplh_DataSelect0_Result : INTEGER :=  6;
CONSTANT tphl_DataSelect0_Result : INTEGER :=  7;
CONSTANT tplh_Data0_Result   : INTEGER :=  8;
CONSTANT tphl_Data0_Result   : INTEGER :=  9;
CONSTANT tplh_Data1_Result   : INTEGER :=  10;
CONSTANT tphl_Data1_Result   : INTEGER :=  11;
CONSTANT tplh_Data2_Result   : INTEGER :=  12;
CONSTANT tphl_Data2_Result   : INTEGER :=  13;
CONSTANT tplh_Data3_Result   : INTEGER :=  14;
CONSTANT tphl_Data3_Result   : INTEGER :=  15;
CONSTANT tplh_Data4_Result   : INTEGER :=  16;
CONSTANT tphl_Data4_Result   : INTEGER :=  17;
CONSTANT tplh_Data5_Result   : INTEGER :=  18;
CONSTANT tphl_Data5_Result   : INTEGER :=  19;
CONSTANT tplh_Data6_Result   : INTEGER :=  20;
CONSTANT tphl_Data6_Result   : INTEGER :=  21;
CONSTANT tplh_Data7_Result   : INTEGER :=  22;
CONSTANT tphl_Data7_Result   : INTEGER :=  23;
CONSTANT tplh_Data8_Result   : INTEGER :=  24;
CONSTANT tphl_Data8_Result   : INTEGER :=  25;
CONSTANT tplh_Data9_Result   : INTEGER :=  26;
CONSTANT tphl_Data9_Result   : INTEGER :=  27;
CONSTANT tplh_Data10_Result  : INTEGER :=  28;
CONSTANT tphl_Data10_Result  : INTEGER :=  29;
CONSTANT tplh_Data11_Result  : INTEGER :=  30;
CONSTANT tphl_Data11_Result  : INTEGER :=  31;
CONSTANT tplh_Data12_Result  : INTEGER :=  32;
CONSTANT tphl_Data12_Result  : INTEGER :=  33;
CONSTANT tplh_Data13_Result  : INTEGER :=  34;
CONSTANT tphl_Data13_Result  : INTEGER :=  35;
CONSTANT tplh_Data14_Result  : INTEGER :=  36;
CONSTANT tphl_Data14_Result  : INTEGER :=  37;
CONSTANT tplh_Data15_Result  : INTEGER :=  38;
CONSTANT tphl_Data15_Result  : INTEGER :=  39;

CONSTANT VitalMUX2delay  : TvalType (tplh_DataSelect0_Result to tphl_Data1_Result);
CONSTANT VitalMUX4delay  : TvalType (tplh_DataSelect1_Result to tphl_Data3_Result);
CONSTANT VitalMUX8delay  : TvalType (tplh_DataSelect2_Result to tphl_Data7_Result);
CONSTANT VitalMUX16delay : TvalType (tplh_DataSelect3_Result to tphl_Data15_Result);
---------------------------------------------------------------
-- DECODER
---------------------------------------------------------------
CONSTANT tplh_Enable_Result : INTEGER := 0;
```

```
CONSTANT tphl_Enable_Result : INTEGER := 1;
CONSTANT tplh_DataN_ResultN : INTEGER := 2;
CONSTANT tphl_DataN_ResultN : INTEGER := 3;

CONSTANT VitalDECODERdelay : TvalType (0 to 3);

---------------------------------------------------------------
-- SCANFF
---------------------------------------------------------------
CONSTANT tplh_CLK_Q          : INTEGER := 0;
CONSTANT tphl_CLK_Q          : INTEGER := 1;
CONSTANT tplh_CLK_Q_N        : INTEGER := 2;
CONSTANT tphl_CLK_Q_N        : INTEGER := 3;
CONSTANT tpwl_CLK            : INTEGER := 4;
CONSTANT tpwh_CLK            : INTEGER := 5;
CONSTANT tperiod_CLK         : INTEGER := 6;
CONSTANT tsetup_DATA0_CLK    : INTEGER := 7;
CONSTANT tsetup_DATA1_CLK    : INTEGER := 8;
CONSTANT thold_DATA0_CLK     : INTEGER := 9;
CONSTANT thold_DATA1_CLK     : INTEGER := 10;
CONSTANT trelease_CLR0_CLK   : INTEGER := 11;
CONSTANT trelease_CLR1_CLK   : INTEGER := 12;
CONSTANT thold_CLR0_CLK      : INTEGER := 13;
CONSTANT thold_CLR1_CLK      : INTEGER := 14;
CONSTANT tphl_CLR_Q          : INTEGER := 15;
CONSTANT tplh_CLR_Q_N        : INTEGER := 16;
CONSTANT trelease_SET0_CLK   : INTEGER := 17;
CONSTANT trelease_SET1_CLK   : INTEGER := 18;
CONSTANT thold_SET0_CLK      : INTEGER := 19;
CONSTANT thold_SET1_CLK      : INTEGER := 20;
CONSTANT tplh_SET_Q          : INTEGER := 21;
CONSTANT tphl_SET_Q_N        : INTEGER := 22;
CONSTANT tsetup_SCAN0_CLK    : INTEGER := 23;
CONSTANT tsetup_SCAN1_CLK    : INTEGER := 24;
CONSTANT thold_SCAN0_CLK     : INTEGER := 25;
CONSTANT thold_SCAN1_CLK     : INTEGER := 26;
CONSTANT tsetup_SCANSEL0_CLK : INTEGER := 27;
CONSTANT tsetup_SCANSEL1_CLK : INTEGER := 28;
CONSTANT thold_SCANSEL0_CLK  : INTEGER := 29;
CONSTANT thold_SCANSEL1_CLK  : INTEGER := 30;



CONSTANT VitalDFFdelay    : TvalType (0 to 10);
CONSTANT VitalDFFCSdelay  : TvalType (0 to 22);
CONSTANT VitalLATCHdelay  : TvalType (0 to 10);
CONSTANT VitalSCANFFdelay : TvalType (0 to 30);

---------------------------------------------------------------
-- JKFF
---------------------------------------------------------------
CONSTANT tsetup_J0_CLK       : INTEGER :=  7;
CONSTANT tsetup_J1_CLK       : INTEGER :=  8;
CONSTANT thold_J0_CLK        : INTEGER :=  9;
CONSTANT thold_J1_CLK        : INTEGER := 10;
CONSTANT tsetup_K0_CLK       : INTEGER := 11;
CONSTANT tsetup_K1_CLK       : INTEGER := 12;
CONSTANT thold_K0_CLK        : INTEGER := 13;
CONSTANT thold_K1_CLK        : INTEGER := 14;
```

```
    CONSTANT VitalJKFFdelay      : TvalType (0 to 14);

    ---------------------------------------------------------------
    -- WIRE DELAY MODEL
    ---------------------------------------------------------------
    CONSTANT VitalWireDelayFanout : INTEGER;
    CONSTANT VitalWireDelay       : TvalType (0 to ((VitalWireDelayFanout*6)-1));

END Tvalpkg_primitives;


PACKAGE BODY Tvalpkg_primitives IS
    CONSTANT VitalGateDelay1 : TvalType(tplh_0 to tphl_0) := (
             tplh_0 => ( 1 ns, 1 ns, 1 ns ),     tphl_0 => ( 1 ns, 1 ns, 1 ns )
    );
    CONSTANT VitalGateDelay2 : TvalType(tplh_0 to tphl_1) := (
             tplh_0 => ( 1 ns, 1 ns, 1 ns ),     tphl_0 => ( 1 ns, 1 ns, 1 ns ),
             tplh_1 => ( 1 ns, 1 ns, 1 ns ),     tphl_1 => ( 1 ns, 1 ns, 1 ns )
    );
    CONSTANT VitalGateDelay4 : TvalType(tplh_0 to tphl_3) := (
             tplh_0 => ( 1 ns, 1 ns, 1 ns ),     tphl_0 => ( 1 ns, 1 ns, 1 ns ),
             tplh_1 => ( 1 ns, 1 ns, 1 ns ),     tphl_1 => ( 1 ns, 1 ns, 1 ns ),
             tplh_2 => ( 1 ns, 1 ns, 1 ns ),     tphl_2 => ( 1 ns, 1 ns, 1 ns ),
             tplh_3 => ( 1 ns, 1 ns, 1 ns ),     tphl_3 => ( 1 ns, 1 ns, 1 ns )
    );
    CONSTANT VitalGateDelay6 : TvalType(tplh_0 to tphl_5) := (
             tplh_0 => ( 1 ns, 1 ns, 1 ns ),     tphl_0 => ( 1 ns, 1 ns, 1 ns ),
             tplh_1 => ( 1 ns, 1 ns, 1 ns ),     tphl_1 => ( 1 ns, 1 ns, 1 ns ),
             tplh_2 => ( 1 ns, 1 ns, 1 ns ),     tphl_2 => ( 1 ns, 1 ns, 1 ns ),
             tplh_3 => ( 1 ns, 1 ns, 1 ns ),     tphl_3 => ( 1 ns, 1 ns, 1 ns ),
             tplh_4 => ( 1 ns, 1 ns, 1 ns ),     tphl_4 => ( 1 ns, 1 ns, 1 ns ),
             tplh_5 => ( 1 ns, 1 ns, 1 ns ),     tphl_5 => ( 1 ns, 1 ns, 1 ns )
    );
    CONSTANT VitalGateDelay8 : TvalType(tplh_0 to tphl_7) := (
             tplh_0 => ( 1 ns, 1 ns, 1 ns ),     tphl_0 => ( 1 ns, 1 ns, 1 ns ),
             tplh_1 => ( 1 ns, 1 ns, 1 ns ),     tphl_1 => ( 1 ns, 1 ns, 1 ns ),
             tplh_2 => ( 1 ns, 1 ns, 1 ns ),     tphl_2 => ( 1 ns, 1 ns, 1 ns ),
             tplh_3 => ( 1 ns, 1 ns, 1 ns ),     tphl_3 => ( 1 ns, 1 ns, 1 ns ),
             tplh_4 => ( 1 ns, 1 ns, 1 ns ),     tphl_4 => ( 1 ns, 1 ns, 1 ns ),
             tplh_5 => ( 1 ns, 1 ns, 1 ns ),     tphl_5 => ( 1 ns, 1 ns, 1 ns ),
             tplh_6 => ( 1 ns, 1 ns, 1 ns ),     tphl_6 => ( 1 ns, 1 ns, 1 ns ),
             tplh_7 => ( 1 ns, 1 ns, 1 ns ),     tphl_7 => ( 1 ns, 1 ns, 1 ns )
    );

    ---------------------------------------------------------------
    -- Three State Buffers and Inverters
    ---------------------------------------------------------------
    CONSTANT VitalBUFFERdelay : TvalType(0 to 5) := (
             tplh_Data_Result   => ( 1 ns, 1 ns, 1 ns ),
             tphl_Data_Result   => ( 1 ns, 1 ns, 1 ns ),
             tplz_Enable_Result => ( 1 ns, 1 ns, 1 ns ),
             tphz_Enable_Result => ( 1 ns, 1 ns, 1 ns ),
             tpzl_Enable_Result => ( 1 ns, 1 ns, 1 ns ),
             tpzh_Enable_Result => ( 1 ns, 1 ns, 1 ns )
    );

    ---------------------------------------------------------------
    -- MUX
    ---------------------------------------------------------------
    CONSTANT VitalMUX2delay : TvalType (tplh_DataSelect0_Result to tphl_Data1_Result) := (
```

```
        tplh_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data1_Result   => ( 1 ns, 1 ns, 1 ns )
);
CONSTANT VitalMUX4delay  : TvalType (tplh_DataSelect1_Result to tphl_Data3_Result) := (
        tplh_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data3_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data3_Result   => ( 1 ns, 1 ns, 1 ns )
);
CONSTANT VitalMUX8delay  : TvalType (tplh_DataSelect2_Result to tphl_Data7_Result) := (
        tplh_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect2_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect2_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data3_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data3_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data4_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data4_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data5_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data5_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data6_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data6_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data7_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data7_Result   => ( 1 ns, 1 ns, 1 ns )
);
CONSTANT VitalMUX16delay : TvalType (tplh_DataSelect3_Result to tphl_Data15_Result) := (
        tplh_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect0_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect1_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect2_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect2_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataSelect3_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataSelect3_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data0_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data1_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data2_Result   => ( 1 ns, 1 ns, 1 ns ),
```

```
        tplh_Data3_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data3_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data4_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data4_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data5_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data5_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data6_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data6_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data7_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data7_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data8_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data8_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data9_Result   => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data9_Result   => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data10_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data10_Result  => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data11_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data11_Result  => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data12_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data12_Result  => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data13_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data13_Result  => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data14_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data14_Result  => ( 1 ns, 1 ns, 1 ns ),
        tplh_Data15_Result  => ( 1 ns, 1 ns, 1 ns ),
        tphl_Data15_Result  => ( 1 ns, 1 ns, 1 ns )
);
---------------------------------------------------------------
-- DECODER
---------------------------------------------------------------
CONSTANT VitalDECODERdelay : TvalType (0 to 3) := (
        tplh_Enable_Result => ( 1 ns, 1 ns, 1 ns ),
        tphl_Enable_Result => ( 1 ns, 1 ns, 1 ns ),
        tplh_DataN_ResultN => ( 1 ns, 1 ns, 1 ns ),
        tphl_DataN_ResultN => ( 1 ns, 1 ns, 1 ns )
);
---------------------------------------------------------------
-- SCANFF
---------------------------------------------------------------
CONSTANT VitalSCANFFdelay : TvalType (0 to 30) := (
        tplh_CLK_Q          => ( 1 ns, 1 ns, 1 ns ),
        tphl_CLK_Q          => ( 1 ns, 1 ns, 1 ns ),
        tplh_CLK_Q_N        => ( 1 ns, 1 ns, 1 ns ),
        tphl_CLK_Q_N        => ( 1 ns, 1 ns, 1 ns ),
        tpwl_CLK            => ( 1 ns, 1 ns, 1 ns ),
        tpwh_CLK            => ( 1 ns, 1 ns, 1 ns ),
        tperiod_CLK         => ( 1 ns, 1 ns, 1 ns ),
        tsetup_DATA0_CLK    => ( 1 ns, 1 ns, 1 ns ),
        tsetup_DATA1_CLK    => ( 1 ns, 1 ns, 1 ns ),
        thold_DATA0_CLK     => ( 1 ns, 1 ns, 1 ns ),
        thold_DATA1_CLK     => ( 1 ns, 1 ns, 1 ns ),
        trelease_CLR0_CLK   => ( 1 ns, 1 ns, 1 ns ),
        trelease_CLR1_CLK   => ( 1 ns, 1 ns, 1 ns ),
        thold_CLR0_CLK      => ( 1 ns, 1 ns, 1 ns ),
        thold_CLR1_CLK      => ( 1 ns, 1 ns, 1 ns ),
        tphl_CLR_Q          => ( 1 ns, 1 ns, 1 ns ),
        tplh_CLR_Q_N        => ( 1 ns, 1 ns, 1 ns ),
        trelease_SET0_CLK   => ( 1 ns, 1 ns, 1 ns ),
        trelease_SET1_CLK   => ( 1 ns, 1 ns, 1 ns ),
        thold_SET0_CLK      => ( 1 ns, 1 ns, 1 ns ),
```

```
            thold_SET1_CLK      => ( 1 ns, 1 ns, 1 ns ),
            tplh_SET_Q          => ( 1 ns, 1 ns, 1 ns ),
            tphl_SET_Q_N        => ( 1 ns, 1 ns, 1 ns ),
            tsetup_SCAN0_CLK    => ( 1 ns, 1 ns, 1 ns ),
            tsetup_SCAN1_CLK    => ( 1 ns, 1 ns, 1 ns ),
            thold_SCAN0_CLK     => ( 1 ns, 1 ns, 1 ns ),
            thold_SCAN1_CLK     => ( 1 ns, 1 ns, 1 ns ),
            tsetup_SCANSEL0_CLK => ( 1 ns, 1 ns, 1 ns ),
            tsetup_SCANSEL1_CLK => ( 1 ns, 1 ns, 1 ns ),
            thold_SCANSEL0_CLK  => ( 1 ns, 1 ns, 1 ns ),
            thold_SCANSEL1_CLK  => ( 1 ns, 1 ns, 1 ns )
  );
    CONSTANT VitalDFFdelay    : TvalType (0 to 10) := VitalSCANFFdelay(0 to 10);
    CONSTANT VitalDFFCSdelay  : TvalType (0 to 22) := VitalSCANFFdelay(0 to 22);
    CONSTANT VitalLATCHdelay  : TvalType (0 to 10) := VitalSCANFFdelay(0 to 10);

    ---------------------------------------------------------------
    -- JK-FLIP-FLOP
    ---------------------------------------------------------------
    CONSTANT VitalJKFFdelay : TvalType (0 to 14) := (
            tplh_CLK_Q          => ( 1 ns, 1 ns, 1 ns ),
            tphl_CLK_Q          => ( 1 ns, 1 ns, 1 ns ),
            tplh_CLK_Q_N        => ( 1 ns, 1 ns, 1 ns ),
            tphl_CLK_Q_N        => ( 1 ns, 1 ns, 1 ns ),
            tpwl_CLK            => ( 1 ns, 1 ns, 1 ns ),
            tpwh_CLK            => ( 1 ns, 1 ns, 1 ns ),
            tperiod_CLK         => ( 1 ns, 1 ns, 1 ns ),
            tsetup_J0_CLK       => ( 1 ns, 1 ns, 1 ns ),
            tsetup_J1_CLK       => ( 1 ns, 1 ns, 1 ns ),
            thold_J0_CLK        => ( 1 ns, 1 ns, 1 ns ),
            thold_J1_CLK        => ( 1 ns, 1 ns, 1 ns ),
            tsetup_K0_CLK       => ( 1 ns, 1 ns, 1 ns ),
            tsetup_K1_CLK       => ( 1 ns, 1 ns, 1 ns ),
            thold_K0_CLK        => ( 1 ns, 1 ns, 1 ns ),
            thold_K1_CLK        => ( 1 ns, 1 ns, 1 ns )
  );

    ---------------------------------------------------------------
    -- WIRE DELAY MODEL
    ---------------------------------------------------------------
    CONSTANT VitalWireDelayFanout : INTEGER := 2;
    CONSTANT VitalWireDelay       : TvalType (0 to ((VitalWireDelayFanout*6)-1)) := (
  others => ( 1 ns, 1 ns, 1 ns )
    );

END Tvalpkg_primitives;
```

14.2 Vital Primitives Coding Example

An example of coding styles used to implement the VITAL_AND gate primitive is provided below.

```
----------------------------------------------------------------------
-- ARCHITECTURAL declaration
----------------------------------------------------------------------
ARCHITECTURE BEHAVIORAL OF VITAL_AND IS
BEGIN
    --------------------------------------------------------
    -- FUNCTIONALITY
    --------------------------------------------------------
    vital : PROCESS
        VARIABLE Answer, AnswerLastValue : std_logic;
        VARIABLE tplh, tphl : TIME;
        VARIABLE last, TvalIndex, PortIndex : INTEGER;
        SUBTYPE DataLastValueType is std_logic_vector(0 to (Size-1));
        VARIABLE DataLastValue : DataLastValueType;
    BEGIN
        assert ((DATA'high - DATA'low) = (Tval'high(1) - Tval'low(1)))
        report "Modeling Error: VITAL_AND (" & InstancePath &
               ")...Insufficient number of timing parameters contrasted to the number of ports"
        severity ERROR;
        Loop
            answer := '1';
            -- Determine the functional answer
            for i in data'range loop
                answer := answer AND data(i);
            end loop;

            -- Choose the correct timing value
            tplh := TIME'HIGH;
            tphl := TIME'HIGH;
            TvalIndex := Tval'low(1);
            PortIndex := DATA'low;  -- start with the first port
            While (PortIndex <= DATA'high) loop
                if (DataLastValue(PortIndex) /= Data(PortIndex)) then
                    tplh := MINIMUM (tplh, Tval(TvalIndex)(TimeMode));
                    TvalIndex := TvalIndex + 1;
                    tphl := MINIMUM (tphl, Tval(TvalIndex)(TimeMode));
                    TvalIndex := TvalIndex + 1;
                else
                    TvalIndex := TvalIndex + 2;
                end if;
                PortIndex := PortIndex + 1;
            end loop;
            -- this minimizes transactions
            if (answer /= AnswerLastValue) then
                Result <= answer after CalcDelay(answer, AnswerLastValue, tplh, tphl );
            end if;
            AnswerLastValue := Answer;
            DataLastValue := Data;
            wait on DATA;
        end loop;

    END PROCESS vital;
END BEHAVIORAL;
```

15.0 Appendix G: Proposed Language Changes

There are a number of changes in the VHDL which could significantly aid in developing models and providing a more concise description of certain classes of designs (i.e. state machines) in addition to aiding in the description and handling of timing information.

NOTE: THE PROPOSED LANGUAGE CHANGES IN THIS SECTION ARE NOT A PART OF THE VITAL STANDARD AT THIS TIME.

The following changes are hereby proposed.

15.1 Definition of a modified PROCESS Design Unit

The current language does not provide a concise syntax for the coding of state machines. Given their pervasive use in common digital logic design, it would seem reasonable to consider language changes which accommodate this technique.

TRUTH TABLES:

The extensions to VHDL required to support this technique include:

- Expansion of the entity class to include the PROCESS
- Simulators would have to recognize that the 1164 '-' represents a don't care semantic and then process the conditional statements accordingly.

```
attribute RECOGNIZE_1164 of mux:PROCESS is TRUE;
mux: PROCESS ( mux, control, dataA, dataB) IS
BEGIN
   mux <=  '1' WHEN (control, dataA, dataB) = "01-" ELSE
           '0' WHEN (control, dataA, dataB) = "00-" ELSE
           '1' WHEN (control, dataA, dataB) = "1-1" ELSE
           '0' WHEN (control, dataA, dataB) = "1-0" ELSE
           '0' WHEN (control, dataA, dataB) = "-00" ELSE
           '1' WHEN (control, dataA, dataB) = "-11" ELSE
           '-';
END PROCESS mux;
```

STATE TABLES:

The extensions to VHDL required to support state tables include:

- understanding of negative, positive, rising, falling and stable edges.
- implicitly defined qualified expressions for 1164 aggregates.
- same extensions as proposed for the Truth Table implementation.

```
dff: PROCESS (q, clock, data) IS
BEGIN
   CASE (clock, d, q) IS
           WHEN (pos_edge, 0, '-') => q <= '0';
           WHEN (pos_edge, 1, '-') => q <= '1';
```

```
            WHEN (any_edge, 1, '-') => q <= '1';

    END CASE;
END PROCESS dff;
```

15.2 Allow dynamic indices in 'Event and 'Last_Value for composite types

The current language restricts the use of 'EVENT and 'LAST_VALUE to static indices of a composite type. For example, in the current language the following is illegal:

```
For i in databus'range loop
   if databus(i)'event then
            do_something;
   endif;
end loop;
```

Under the current language, a developer must declare and manage a copy of the previous value of the bus, perform an equality test on each subelement to determine if the value changes since the last invocation of the model.

15.3 Glitch Detection and 'X' generation

The syntax of the proposed change is as follows:

```
target <= waveform_element
   [ GLITCH glitch_value [ AFTER time_expression][ REPORT string]];
```

NOTE: Racal-Redac [Fazakerly & Sissler] has made this excellent suggestion. Further details of this proposal will be provided in the next release of this document.

15.4 Need for a 'PREVIOUS_EVENT attribute

It is often useful to determine the time since the last event of a signal. However, if the signal has just become active, then S'last_event == 0 ns. Current practice requires the use of S'DELAYED'LAST_EVENT. However, the use of this technique is quite costly since it creates a delta delayed copy of the signal S.

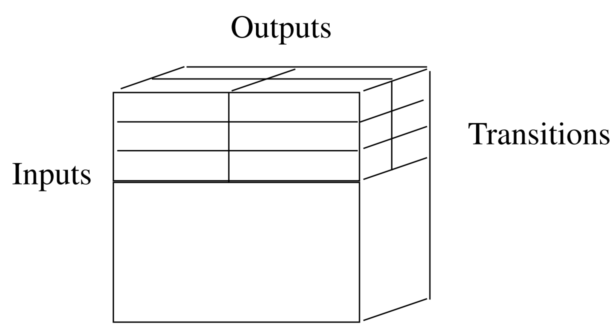
A proposed addition of a S'PREVIOUS_EVENT attribute could provide the equivalent value at a much lower runtime cost. {Fazakerly & Sissler]

15.5 Implicit Timing Model

It is recognized that a need exists to standardize the way timing is represented for ASIC modeling while at the same time, allowing leading-edge vendors the flexibility they may require to extend the specific methods by which timing information is derived or handled within a model.

The timing characteristics of a particular device within a structural model is essentially a function of two environmental conditions (temperature and voltage) and three device-specific characteristics (intrinsic delay, net loading, and layout). The intrinsic delay of the device is the time it takes to propagate a value through the device. This value is characterized by an ASIC vendor based on the process technology, cell complexity, and design techniques. This value may be different for distinct paths through the device often referred to as pin-to-pin timing. The net loading is a function of the number of devices to which an output pin is connected. The layout contributes timing characteristics based on the wire lengths and impedance. Furthermore, the state of the device can alter the timing, especially the pin-to-pin timing characteristics. In particular, the delay through the device can change based on the value transition of an active input pin (i.e., rising and falling times can be different).

The IMPLICIT timing model simplifies this general timing view by collapsing these various timing effects into a three-dimensional matrix of timing values describing the pin-to-pin timing on a device. This is achieved by folding the delays introduced by net loading and layout into the delays on the inputs and outputs of the device itself. This is a common paradigm which is typical of ASIC models today. The dimensions of the matrix are inputs, outputs, and state transitions.



Each cell is filled in using the following equation:

cell_{IOT} = Layout-delay$_I$ + IOPath Delay$_{IOT}$ + Load-delay$_O$
for a given input (I), output (O), and transition (T).

The layout-delay is taken from the wire delay on the wire which is connected to the device at the given input pin. The IO path delay is specified for the device and may be different for the different transitions of the input pin value. The load delay is taken from the load delay for the given output pin. It is possible to additionally represent the cell by an expression based on the state of multiple pins or internal signals.

Under this proposal this timing semantic is applied to the device by a tool which is processing the circuit. The device itself is described by its functional behavior using VHDL. When a tool is processing the device (e.g., simulating the device) it constructs this three-dimensional matrix and associates it with the functional description of the device. An alternative to this approach would be to implement this matrix model directly in the description of the device. The choice to separate the functional model from the timing model has a number of significant advantages including:

- the ability to apply different timing characteristics to the same device based on different environmental factors (i.e., temperature and voltage) or from different usages within the design process (e.g., pre-layout estimation, post-layout estimation, etc.),
- the ability to optimize the matrix model when some timing characteristics are not specified (e.g., if no transition-dependent delays are specified the matrix can be collapsed into two dimensions),
- the ability to optimize the execution of the device by removing all event scheduling within the device itself, and
- the capacity to use matrix cross products to further combine functional models and reduce event scheduling across collections of devices.

In an attempt to satisfy these requirements along with those of modeling and simulation efficiency, the model chosen is based on an implicit matrix which defines the timing relationships between all inputs and outputs of a model. This implied matrix represents the same information as the traditional set of generic parameters seen in many VHDL models but avoids the overhead of explicitly storing and processing the data and source text. It provides a simple mechanism for simulators and related design tools to optimize data handling.

15.5.1 Implicit Timing Model Restrictions

VITAL macrocell models are intended for specific applications requiring high speed simulation. In order to achieve this goal the models must not contain explicit delays nor reference global signals.

15.5.2 Implicit Timing Model Overview

An implicit timing model is a legal VHDL architecture with the following restrictions:

- the model contains no explicit delays i.e. no AFTER clauses

A VHDL simulator which does not support the implicit model may process the model the same way a non-implicit model is processed. It would function as a UNIT delay model.

In the absence of a timing file, the implicit timing model will be simulated as a UNIT delay model. If a corresponding timing file is found, the timing characteristics of the model will be defined by that file.

The delay through the model will be determined by the relations between input and outputs of the model as specified by the timing file. These relations may be simple constants for the device, depend on the input which had the event that triggered the process or be dependent on a more complex function of the states of the inputs. The appropriate relations are determined by the timing file. The model need not change.

For networks consisting purely of implicit models, similar relations may be used for wire delays between the models.

15.5.3 Implicit Timing Model Characterization

The implicit timing model is characterized in terms of its input and output ports. For each input/output port pair there is an implied timing relationship defined. The major difference between this style and the more general VHDL modeling style is that in an implicit timing model, data need only be supplied by the timing file for relations which actually affect simulation, while in a more general style, place holders and at least default data most be supplied for all relations unless the more general models are changed each time the dependencies change.

For example, the VITAL modeling specification shows a general VHDL model which has been optimized to reduce generic parameters and isolate timing data in separate packages. This model represents the results of significant effort, expertise, and experience in the optimization of VHDL models. However in spite of this, the overhead is significant especially for relatively simple devices which tend to make up the bulk of ASIC libraries.

Since the timing packages which contain this data are essentially text representations of the cross product of inputs, outputs, and state transitions coupled with an indexing schema this structure may be implied without being explicitly stated. This approach saves significant amounts of storage and processing time since in reality the number of relations used in any particular model is relatively sparse compared to the full matrix.

15.5.4 Implicit Timing Model Format

The model written in an implicit format is similar but does not require timing mode parameters or references to a timing package. The information required is supplied by the external timing file.

```
entity FF is
  port ( D, CLK, CLR, SET, SCN: in std_logic;
           Q, Q_N, ZOUT: out std_logic;
end FF;

-- example VITAL architecture
architecture EXAMPLE of FF is
  attribute VITAL of EXAMPLE: architecture is TRUE;
begin
  CELL1: process (CLK, CLR, SET)
          ...
          Q <= VAL1;
  end process;

  CELL2: process(CLK, CLR, SET)
          ....
          Q <= VAL2;
  end process;
end EXAMPLE
```

TABLE 3. IMPLICIT DELAY TABLE FOR FF

	Q	Q_N	ZOUT
D	2	3	3
CLK	3	4	4
CLR	3	4	4
SET	4	5	5
SCN	4	5	5

The implicit delay table is left simple since it is just intended to illustrate the concept. The delay across any of the cells in the model is determined by the input on which the event which kick the process occurred and the output signal to which it propagates. It will be illustrated in a later section how complex delays and interactions can be handled.

15.5.5 Implicit Timing Process Model

In a general VHDL model there is no assumption made that the signal which kicks off a process is relevant to the timing characteristic of the process. It is assumed that the timing will be embedded in the delays associated with the functionality of the model. In order to recover the information about the initiating signal, special code must be included in the model to detect the event on that signal and some form of case statement or conditional statement must be used to decode the causality of the process initiation. In general this is a sensible semantic since there is no a priori reason to believe, in the VHDL process model, that the signal which kicks off a process will be any more important than any other signal. A simulator will generally not make a special effort to “remember” this signal nor to optimize execution of code which attempts to determine it.

In the implicit timing model, it is assumed that timing characteristics of a model are tightly coupled to the initiating signal since this is a generally accepted paradigm for ASIC cell design. The advantage of this focus is that the simulator “knows” to “remember” the initiating signal and that code for detecting this information is unnecessary. This allows the simulator to reduce the number of processes and signals since those generated specifically for detecting the active input are eliminated. Furthermore, by eliminating internal delays from the model and disallowing reference to global signals, the job of event scheduling is greatly reduced in complexity.

Generally the biggest performance bottleneck in an event driven simulator is event scheduling. This is particularly true for VHDL simulators because of the generality of the process and signal update semantics. The few limitations imposed on an implicit timing model together with the implied path delay model will allow simulators to perform simple optimizations which will improve performance by orders of magnitude. Similar performance improvements are already available in gate level simulators which make similar assumptions about a specific underlying timing model.

An example is shown of an ASIC model for a logic gate contrasts the behavioral representation with its extra signals and processes with a parallel structural model which uses a built in primitive and external timing files for back annotation data. This practise shows the industry direction with respect to modeling styles for ASICs i.e. use an behavioral model for illustrative purposes but use a built in structural model for simulation performance and back annotation.

16.0 Appendix H: SDF to VITAL Timing Package Mapping

SDF syntax	Equivalent VHDL Implementation
<delay_file> ::=(DELAYFILE <sdf_version> <design_name> <date> <vendor> <program_name> <program_version> <hierarchy_divider> <voltage> <process> <temperature> <time_scale> <cell>?)	PACKAGE InstanceData_<design_name> IS --- -- SDF Version: <sdf_version_qstring> -- Design: <design_name_qstring> -- Date: <date_qstring> -- Vendor: <vendor_qstring> -- Program Name: <program_name_qstring> -- Program Version: <program_version_qstring> -- Voltage: <voltage_rexpression> -- Process: <process_qstring> -- Temperature: <temperature_rexpression> Note: also used to set the units of time in the time aggreagte within the package body -- TimeScale: <timescale_tsvalue <timeunit>>
<cell> ::=(CELL <celltype> <instancename> <timing_def>?)	CONSTANT T<cellname>_<instancename> : TvalType (1 to nparameters);
<timing_def> ::=(DELAY <deltype>?)	This statement starts the creation of the deferred constant's aggregate value in the corresponding package body.
||= (TIMINGCHECK <tc_def>?)	This statement has no particular effect.
||= (INCLUDE <fpath>)	Ignored
<deltype> ::=(ABSOLUTE <absvals>?)	This statement has no particular effect
||= (INCREMENT <incvals>?)	This statement has no particular effect
||= (USERDEF (<qstring>) (<qstring>?))	Ignored
||= (PATHPULSE {<port_path> <port_path>} ((<expression>)) {(<expression>)})	Ignored
||= (GLOBALPATHPULSE {<port_path> <port_path>} ((<expression>)) {(<expression>)})	Ignored
<absvals> ::=(IOPATH <port_spec> <port_path> <value>)	This statement creates timing value constant aggregate values with prefix names corresponding to their position in the 2,3 or 6 value SDF timing set. An example follows: <tprefix>_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>), Example: tplh_CLK_Q => (10 ns, 20 ns, 30 ns),

<table>
<tr><td>||= (COND <cond_port_expression>(IOPATH <port_spec> <port_path> <value>))</td><td>Conditions are ignored. The model VHDL code defines the conditions.</td></tr>
<tr><td>||= (PORT <port_path> <value>)</td><td>This statement causes wire delay models to be created and potentially inserted in the VHDL netlist.</td></tr>
<tr><td>||= (INTERCONNECT <port_instance> <port_instance> <value>)</td><td>Ignored</td></tr>
<tr><td>||= (NETDELAY <name> <value>)</td><td>This statement causes wire delay models to be created and potentially inserted in the VHDL netlist.</td></tr>
<tr><td>||= (DEVICE {<port_path>} <value>)</td><td>Ignored</td></tr>
</table>

<tcheck>::=

<table>
<tr><td>(SETUP <port_path> <port_spec> (<tc_rvalue>))</td><td>tsetup_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||= (HOLD <port_path> <port_spec> (<tc_rvalue>))</td><td>thold_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(SETUPHOLD <port_path> <port_spec> (<tc_rvalue>) (<tc_rvalue>))</td><td>tsetup_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),
thold_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(RECOVERY<port_spec> <port_spec> (<tc_rvalue>))</td><td>trecovery_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(SKEW <port_spec> <port_spec> (<tc_rvalue>))</td><td>tskew_<PortA>_<PortB> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(WIDTH <port_spec> (<tc_value>))</td><td>tpwl_<port> => (<tmin>,<ttyp>,<tmax>),
tpwh_<port> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(PERIOD <port_spec> (<tc_value>))</td><td>tperiod_<InputPort>_<OutputPort> => (<tmin>,<ttyp>,<tmax>),</td></tr>
<tr><td>||=(NOCHANGE <port_spec> <port_path> (<tc_rvalue>))</td><td>Ignored</td></tr>
<tr><td>||=(PATHCONSTRAINT <port_instance> <port_instance>? (<tc_rvalue>))</td><td>Ignored</td></tr>
<tr><td>||=(SUM (<port_instance> <port_instance) <(<port_instance> <port_instance>)>? (<tc_rvalue>))</td><td>Ignored</td></tr>
<tr><td>||= (DIFF (<port_instance> <port_instance>) (<port_instance> <port_instance>) <tc_rvalue>))</td><td>Ignored</td></tr>
<tr><td>||= (SKEWCONSTRAINT <port_spec> (<tc_rvalue>))</td><td>Ignored</td></tr>
</table>